govready-q Documentation

Release 0.8.6

Josh Tauberer, Greg Elin

Oct 16, 2019

CONTENTS:

Note: This documentation set is part of GovReady v0.9.0. In places, it may still refer to v0.8.6 software features rather than v0.9.0 features. Thank you for your patience while we update the documentation to apply to the v0.9.0 software. *Pull requests are very welcome!*

The GovReady-Q Compliance Server is an open source GRC platform for highly automated, user-friendly, self-service compliance assessments and documentation. It's perfect for DevSecOps.

GovReady-Q solves the painful compliance bottleneck of needing months to authorize applications that deploy and redeploy in minutes.

The code is open source, and available on GitHub.

Attention: GovReady-Q is in Beta. Suggested for DevSecOps early adopters needing Compliance-as-Code.

CHAPTER

ONE

ABOUT GOVREADY-Q

1.1 Why GovReady-Q?

Everything about developing and deploying software is accelerating. . . except for compliance. Why? Because:

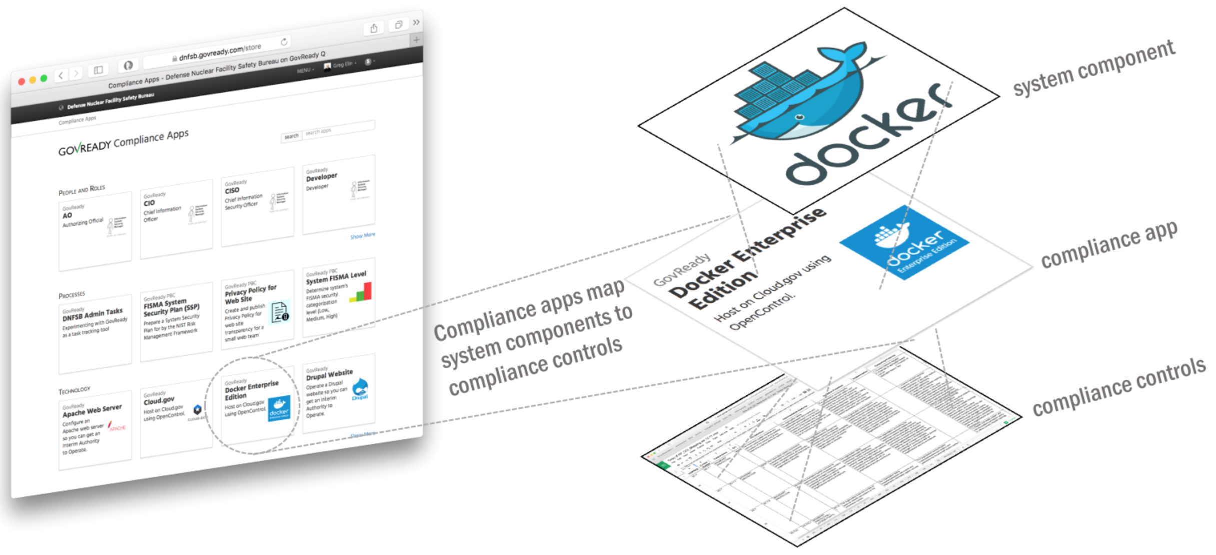
- maintaining written documentation is too slow,
- pondering how jargon-laden control guidance applies is too hard,
- there's little reuse, and no compliance documentation supply chain.

To stop needing months to authorize systems that deploy in minutes, assessments and authorizations need to be assembled from vetted, pre-fabricated components sourced from the same software supply chain with which we assemble applications.

1.2 How GovReady-Q Accelerates Compliance

GovReady accelerates compliance through component-centric guidance, pre-written documentation, and collaboration.

When you use or install GovReady-Q, you gain access to a marketplace of small, self-service compliance apps written by peers and vendors that map system components to security controls and guide you step-by-step through assessments and documentation.


GOVREADY Compliance Apps
system component
compliance app
compliance controls
Compliance apps map system components to compliance controls
GovReady Docker Enterprise Edition
Host on Cloud.gov using OpenControl.

As you and your teammates collaboratively answer questions, the compliance apps work with GovReady-Q to store your data in a relational database and automagically generate and maintain your compliance artifacts for auditors.

GovReady-Q's contribution to Compliance-as-Code is the data abstractions for shareable, reusable, and customizable packages—*Compliance Apps*—to map the relationship between a system component and a set of controls. The approach is innovative, yet familiar. Compliance Apps:

- enable a hub/marketplace for community contributions;
- extend inherited controls model to each system component;
- enable modern, user-friendly experiences;
- support agile, iterative workflows.

Attention: GovReady-Q software is "Beta" software best suited for early adopters needing faster compliance for DevSecOps.

1.3 GovReady-Q Philosophy

Compliance is not security. Compliance scales security.

Compliance is a technique humans have developed for enabling trust in systems that are too large and complex for individuals to assess trustworthiness. Compliance scales participation, attestation and verification of recommended practices.

- We love security and innovation and believe they enable each other.
- We believe security and compliance are standard, not premium add-ons.
- We view compliance as a by-product of a well-instrumented process.
- We value ease-of-use to increase adoption.
- We value automation to increase consistency.
- We see virtualization and DevOps enabling massive gains in security and compliance.

1.4 GovReady-Q Features

- Easy-to-use, beautiful questionnaires
- Jargon-free approach to security controls and compliance
- Step-by-step guidance through assessments
- Compliance-as-Code approach to creating compliance documentation
- Discuss questions and answers in the tool instead of in email
- Support for rich, clear multi-media and multi-channel communication
- RESTful Automation API to integrate with DevOps pipeline and existing agents
- Innovative, reusable "Compliance Apps" model
- Friendly Open Source license so you can start now

1.5 Using Hosted GovReady-Q

There's nothing to install. Q.GovReady.com is the hosted version of GovReady-Q.

1. Visit Q.GovReady.com
2. Fill out the form "About your organization" and "About you" to create your account
3. Don't worry about the Service Levels – everything's available to everyone during the Beta phase
4. We'll contact you to help you get started

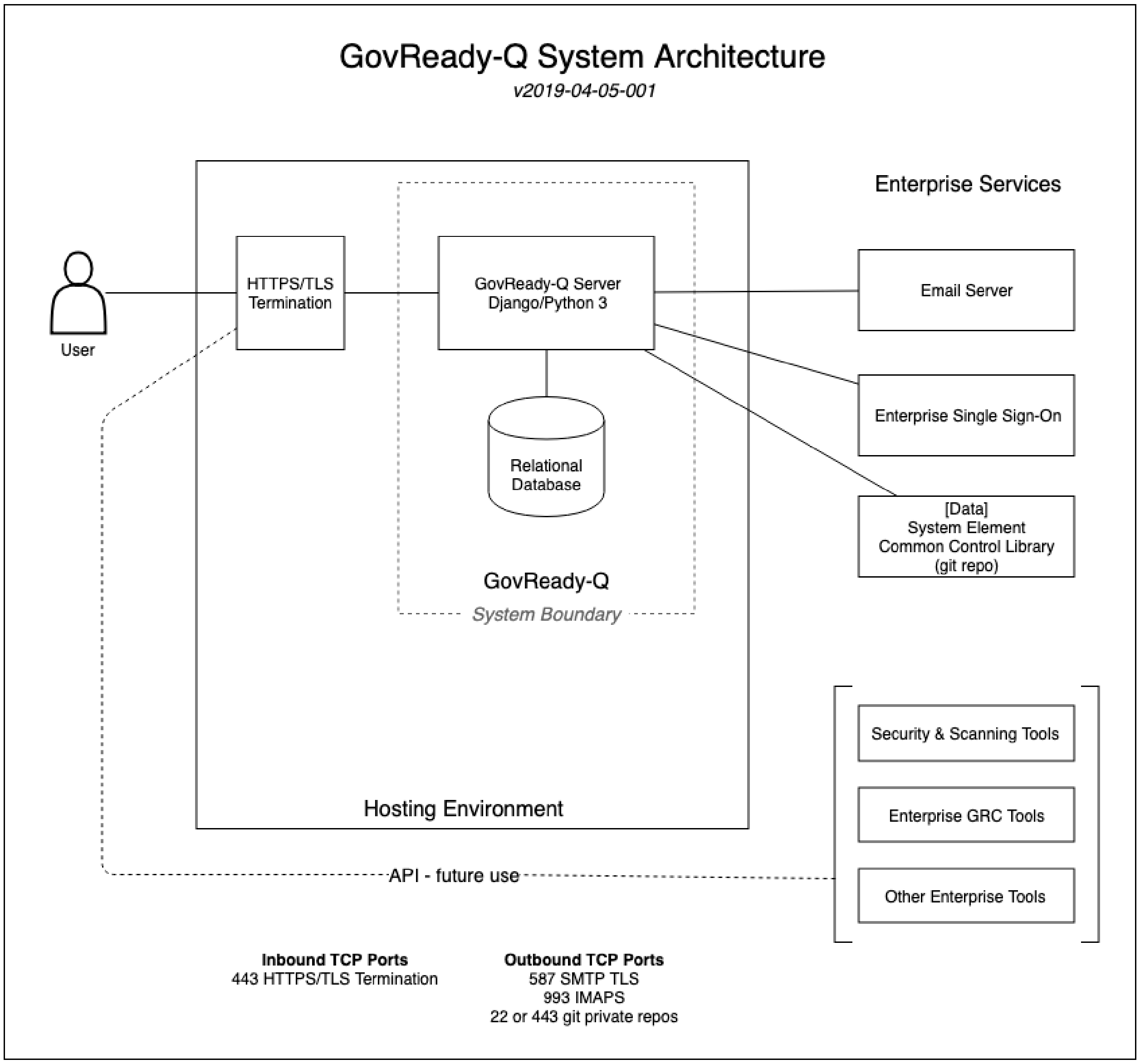
Attention: We will help you get up and running during the current Beta phase of the project while we make getting started easier.

The hosted version is an excellent solution if have one project/system you are trying to get through NIST SP 800-53 or NIST SP 800-171 compliance, or you are have just trying to pull together a few specific compliance documents like your Privacy Policy or Rules of Behavior. The hosted service is operated by GovReady PBC, the company behind GovReady-Q Compliance Server.

If you have questions about the hosted version, email .

1.5.1 System Architecture

The following diagram depicts a generic, high-level system architecture GovReady-Q deployment including external ports and protocols. Architectures vary depending on redundancy requirements, use of containers, etc.



1.6 Downloading GovReady-Q

Downloading	Where
Current Release on Docker	https://hub.docker.com/r/govready/govready-q/
Nightly Build on Docker	https://hub.docker.com/r/govready/govready-q-nightly/
Clone the GitHub repo	https://github.com/govready/govready-q

1.7 Installing GovReady-Q

1.8 Finding Compliance Apps

Compliance Apps are GovReady-Q modular, shareable, reusable, data packages mapping the relationship between system components and security controls. See Understanding Compliance Apps for a more detailed description.

1.8.1 For Hosted Version

When using the Hosted Version of GovReady-Q, GovReady PBC manages the Compliance Apps available to your organization. Send email to to request changes.

1.8.2 For Local Installs

The docker and downloaded version of GovReady-Q automatically loads a small set of example Compliance Apps. Compliance Apps are published in collections known as "AppSources" (e.g., repos). Here are a few:

- https://github.com/GovReady/govready-apps-dev

You can can show and hide compliance apps from the Django administration page at `main.localhost:8000/admin/guidedmodules/appsource/`.

1.8.3 Creating Your Own Compliance Apps

To get started writing your own Compliance Apps see: Creating Compliance Apps.

1.9 Documentation

The official GovReady-Q documentation is maintained at govready-q.readthedocs.io.

1.10 Support

Commercial support for GovReady-Q is provided by GovReady PBC. Email .

Sign up for Security Notifications email list at GovReady Security Alerts.

1.11 Reporting Bugs & Issues

Please file bug reports on our GitHub issue. When reporting a bug, please include as much information as possible. This includes:

- Install type: Hosted, Local, Docker, etc
- URL
- Action taken
- Expected result
- Actual result
- Screenshot (if relevant)

1.12 License / Credits

This repository is licensed under the GNU GPL v3.

- Emoji icons by http://emojione.com/developers/.
- Generic server icon by Stock Image Folio from Noun Project.

1.13 About GovReady PBC

GovReady PBC is a Public Benefit Corporation whose mission is to lower the cost of innovation in digital services to citizens. GovReady's innovative self-service IT compliance tool GovReady-Q was developed as part of an R&D contract to automate and lower the cost of cyber security compliance from the Department of Homeland Security, Science and Technology Directorate, Cyber Security Division. GovReady PBC is based in the greater Washington, DC metro area.

CHAPTER

TWO

DEPLOYING GOVREADY-Q

2.1 System Requirements

GovReady-Q is a Python 3.6 and higher, Django 2.x application with a relational database back-end.

2.1.1 Software Requirements

Required Software Packages (partial list)
(GovReady-Q application)
Python 3.6
Django 2.2
Jinja 2.x
uwsgi 2.x
unzip
graphviz
pandoc
Wkhtmltopdf
Git 2.x
supervisor

Supported Databases
Postgres 9.4 (psycopg2 2.7.5 adapter)
Mysql 7.6 and higher (mysqlclient 1.3.12 adapter)
SQLite 3.x

Recommended Database
Postgres 9.4 (psycopg2 2.7.5 adapter)

SMTP Mail Server (for sending email notifications and receiving comments via email)
Any SMTP mail server (MTA) supporting STARTTLS connections.

For a more detailed list of software dependencies and requirements see:

- https://github.com/GovReady/govready-q/blob/master/requirements.in
- https://github.com/GovReady/govready-q/blob/master/requirements.txt

- https://github.com/GovReady/govready-q/blob/master/requirements_mysql.in
- https://github.com/GovReady/govready-q/blob/master/requirements_mysql.txt
- https://github.com/GovReady/govready-q/blob/master/Vagrantfile

2.1.2 Hardware Requirements

Minimum Hardware
Single server to host both multi-tenant GovReady-Q application and Database
Linux-compatible hardware
2GB RAM
10 GB storage (for database)

Recommended Hardware
2 servers: 1 for multi-tenant GovReady-Q application; 1 for Database Server
Linux-compatible hardware (64 bit architecture; FIPS-140-2 validated cryptographic module)
8GB RAM for each server
100 GB storage (for database server)

2.2 Installing GovReady-Q

Click one of the tab belows to see quickstart for indicated platform.

Docker

Installing with Docker

Make sure you first install Docker (https://docs.docker.com/engine/installation/) and, if appropriate, grant non-root users access to run Docker containers (https://docs.docker.com/engine/installation/linux/linux-postinstall/#manage-docker-as-a-non-root-user) (or else use *sudo* when invoking Docker below).

Start

```
# Run the docker container in detached mode
docker container run --name govready-q --detach -p 8000:8000 govready/govready-q

# Create admin account and organization data
docker container exec -it govready-q first_run

# Stop, start container (when needed)
docker container stop govready-q
docker container start govready-q

# View logs - useful if site does not appear
docker container logs govready-q

# To destroy the container and all user data entered into Q
docker container rm -f govready-q
```

Visit your GovReady-Q site in your web browser at:

http://localhost:8000/

macOS

Installing on macOS

GovReady-Q calls requires Python 3.6 or higher to run and several Unix packages to provide full functionality. Install the Homebrew package manager (https://brew.sh) to easily install Unix packages on macOS. Homebrew will install all packages in your userspace and not change native macOS Python or other libraries.

```
# install Homebrew package manager
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Now install Python3 and the required Unix packages.

```
# install dependencies using brew
brew install python3

# install other packages:
brew install unzip graphviz pandoc selenium-server-standalone
brew cask install wkhtmltopdf
```

Installing GovReady-Q

Clone GovReady-Q source code and install.

```
# clone GovReady-Q
git clone https://github.com/govready/govready-q
cd govready-q

# install Python 3 packages
pip3 install --user -r requirements.txt

# install Bootstrap and other vendor resources locally
./fetch-vendor-resources.sh
```

Run the final setup commands to initialize a local Sqlite3 database in *local/db.sqlite* to make sure everything is OK so far:

```
# run database migrations (sqlite lite database used by default)
python3 manage.py migrate

# load a few critical modules
python3 manage.py load_modules

# create superuser with initial account
python3 manage.py first_run
```

Start GovReady-Q

```
# run the server
python3 manage.py runserver
```

Visit your GovReady-Q site in your web browser at:

http://localhost:8000/

RHEL/CentOS 7

Installing on RHEL/CentOS 7

Instructions applicable RHEL 7, CentOS 7 and Amazon Linux 2.

GovReady-Q calls requires Python 3.6 or higher to run and several Linux packages to provide full functionality.

```
# if necessary, enable EPEL and IUS repositories
rpm -i https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm https://
↪rhel7.iuscommunity.org/ius-release.rpm

# install dependencies
sudo yum install \
unzip python36-pip python36-devel \
graphviz \
pandoc xorg-x11-server-Xvfb wkhtmltopdf \

# optional install gcc to build the uWSGI Python package.
sudo yum install gcc

# optional insall of postgress and/or mysql
sudo yum install postgresql mysql-devel
```

GovReady-Q calls out to *git* to fetch apps from git repositories, but that requires git version 2 or later because of the use of the GIT_SSH_COMMAND environment variable. RHEL stock git is version 1. Switch it to version 2+ by using the IUS package:

```
# if necessary, remove any git currently installed
yum remove git
# install git2u
yum install git2u
```

Installing GovReady-Q

Clone GovReady-Q source code and install.

```
# clone GovReady-Q
git clone https://github.com/govready/govready-q
cd govready-q

# install Python 3 packages
pip3 install --user -r requirements.txt

# install Bootstrap and other vendor resources locally
./fetch-vendor-resources.sh
```

Run the final setup commands to initialize a local Sqlite3 database in *local/db.sqlite* to make sure everything is OK so far:

```
# run database migrations (sqlite lite database used by default)
python3 manage.py migrate

# load a few critical modules
python3 manage.py load_modules

# create superuser with initial account
python3 manage.py first_run
```

Start GovReady-Q

```
# run the server
python3 manage.py runserver
```

Visit your GovReady-Q site in your web browser at:

> http://localhost:8000/

Ubuntu 16.04

Installing on Ubuntu

Instructions provide basic guidance on setting up GovReady-Q on an Ubuntu 16.04 server with Nginx. These commands should be run from the root directory of the GovReady-Q code repository.

GovReady-Q calls requires Python 3.6 or higher to run and several Linux packages to provide full functionality.

```
# upgrade apt-get
apt-get update && apt-get upgrade -y

# install dependencies
apt-get install -y \
  unzip \
  python3 python-virtualenvpython3-pip \
  python3-yaml \
  nginx uwsgi-plugin-python3supervisor \
  memcached \
  graphviz

# optional install gcc to build the uWSGI Python package.
sudo yum install gcc

# optional insall of postgress and/or mysql
apt-get install -y postgresql mysql-devel
```

Installing GovReady-Q

Clone GovReady-Q source code and install.

```
# clone GovReady-Q
git clone https://github.com/govready/govready-q
cd govready-q

# install Python 3 packages
```

(continues on next page)

(continued from previous page)

```
pip3 install --user -r requirements.txt

# install Bootstrap and other vendor resources locally
./fetch-vendor-resources.sh
```

Run the final setup commands to initialize a local Sqlite3 database in *local/db.sqlite* to make sure everything is OK so far:

```
# run database migrations (sqlite lite database used by default)
python3 manage.py migrate

# load a few critical modules
python3 manage.py load_modules

# create superuser with initial account
python3 manage.py first_run
```

Start GovReady-Q

```
# run the server
python3 manage.py runserver
```

Visit your GovReady-Q site in your web browser at:

> http://localhost:8000/

Windows

Installing on Windows (with Docker)

GovReady-Q can only be installed on Windows using Docker.

Make sure you first install Docker (https://docs.docker.com/docker-for-windows/install/).

Start

```
# Run the docker container in detached mode
docker container run --name govready-q --detach -p 8000:8000 govready/govready-q

# Create admin account and organization data
docker container exec -it govready-q first_run

# Stop, start container
docker container stop govready-q
docker container start govready-q

# View logs - useful if site does not appear
docker container logs govready-q

# To destroy the container and all user data entered into Q
docker container rm -f govready-q
```

Visit your GovReady-Q site in your web browser at:

http://localhost:8000/

Additional deployment details and configuration options are documented for each platform.

2.2.1 Deploying with Docker

Container	Where
Current Release on Docker	https://hub.docker.com/r/govready/govready-q/
Release 0.9.0.dev on Docker	https://hub.docker.com/r/govready/govready-0.9.0.dev/
Nightly Build on Docker	https://hub.docker.com/r/govready/govready-q-nightly/

Quickstart

Docker

Installing with Docker

Make sure you first install Docker (https://docs.docker.com/engine/installation/) and, if appropriate, grant non-root users access to run Docker containers (https://docs.docker.com/engine/installation/linux/linux-postinstall/#manage-docker-as-a-non-root-user) (or else use *sudo* when invoking Docker below).

Start

```
# Run the docker container in detached mode
docker container run --name govready-q --detach -p 8000:8000 govready/govready-q

# Create admin account and organization data
docker container exec -it govready-q first_run

# Stop, start container (when needed)
docker container stop govready-q
docker container start govready-q

# View logs - useful if site does not appear
docker container logs govready-q

# To destroy the container and all user data entered into Q
docker container rm -f govready-q
```

Visit your GovReady-Q site in your web browser at:

http://localhost:8000/

Additional Details

Notes and Common Issues

Your GovReady-Q site will not load immediately, as GovReady-Q initializes your database for the first time. Wait for the site to become available.

Because of HTTP Host header checking, you must use `localhost` to access the site, or another hostname if configured using the `--address` option documented below.

If the site does not come up, check the container logs for an error message:

```
docker container logs govready-q
```

The GovReady-Q default SQLite database created within a Docker container exists only for the duration of the container's lifetime. The database will persist between `docker container stop`/`docker container start` commands, but when the container is removed from Docker (i.e. using `docker container rm`) the database will be destroyed. See the *Persistent database* section below for connecting to a database outside of the container for production data.

The default Govready-Q instance cannot send email or receive comment replies until it is configured to use a transactional mail provider like Mailgun – see below.

The default Govready-Q instance is configured to non-debug mode (Django `DEBUG=false`), which is the recommended setting for a public website. The instance can be set to debug mode at runtime – see below.

Advanced configuration of GovReady-Q inside Docker

For more complex setups, using our run script instead will be easier:

```
wget https://raw.githubusercontent.com/GovReady/govready-q/master/deployment/docker/
↪docker_container_run.sh
chmod +x docker_container_run.sh
./docker_container_run.sh
```

Advanced container options can be set with command-line arguments to our container run script:

```
./docker_container_run.sh ...GovReady-Q arguments... -- ...Docker arguments...
```

Changing the hostname and port

The public address (as users see it)

The container will run at `localhost:8000` by default, it will only be accessible from the host machine, and because of HTTP Host header checking you must visit GovReady-Q using the same hostname it is configured to run at (so, with default settings, visiting `127.0.0.1` instead of `localhost` will result in an error).

You may change the hostname and port of the GovReady-Q server using:

```
./docker_container_run.sh --address q.mydomain.com:80
```

If the Docker container is behind a proxy, then `--address` specifies the public address that end-users will use to access GovReady-Q. This may differ from the address and port that the container is accessed at on your organization's network, which is set using `--bind`.

Add `--https` if end users will access GovReady-Q with https: URLs. This must be done through a proxy that accepts HTTPS connections and passes the requests using HTTP to the Docker container. See the `HTTPS` environment variable, below.

The address that the container is bound to

Use `--bind IP:PORT` to control how the listening socket is created on the host machine. The default value of `--bind` is `127.0.0.1` and the port from `--address`, or `127.0.0.1:8000` if `--address` isn't given. If the host machine is behind a proxy, use `--bind` to control the network interface and port that Docker will forward to the GovReady-Q container.

```
./docker_container_run.sh --bind 10.0.0.5:6543
```

Persistent database

In a production environment it is important to have GovReady-Q connect to a persistent database instead of the database stored inside the container, which will be destroyed when the container is destroyed. There are two methods for connecting to a persistent database.

Sqlite file

You can use a Sqlite file stored on the host machine:

```
./docker_container_run.sh --sqlitedb /path/to/govready-q-database.sqlite
```

You must specify an absolute path. The path is mounted using a Docker bind mount into the container filesystem.

The file must be readable and writable by the container process, which is running as user 1000/group 1000. Although the container is running as a user isolated from the host environment, filesystem permissions for mounted files are based on comparing the raw user/group IDs of the file's owner/group on the host to the raw user/group ID of the process running in the container. Consider granting user 1000 read/write permission to the database using ACLs:

```
setfacl -m u:1000:rw /path/to/govready-q-database.sqlite
```

Of course, do not do this if the host machine has a user 1000 that you do not trust.

Remote database

You can also connect to a database running on a remote system accessible to the Docker container.

For instance, you might run a second Docker container holding a Postgres server.

```
DBPASSWORD=mysecretpassword
docker container run --name govready-q-db -e POSTGRES_PASSWORD=$DBPASSWORD -d postgres
DBHOST=$(docker container inspect govready-q-db | jq -r .[0].NetworkSettings.
↪IPAddress)
DBUSER=postgres
DBDATABASE=postgres
```

(This example uses `jq`, a JSON parsing tool, to extract the IP address of the database container. You can install `jq` or just set `DBHOST` manually by looking for the IP address in `docker container inspect govready-q-db`.)

Start the GovReady-Q container with the argument:

```
./docker_container_run.sh --dburl postgres://$DBUSER:$DBPASSWORD@$DBHOST/$DBDATABASE
```

where `$DBHOST` is the hostname of the database server, `$DBDATABASE` is the name of the database, and `$DBUSER` and `$DBPASSWORD` are the credentials for the database.

You can also use a MySQL or MariaDB server using the syntax `mysql://USER:PASSWORD@HOST:PORT/NAME`.

Configuring email

GovReady-Q sends outbound emails for notifications about invitations and discussions. It also receives inbound emails — replies to discussion notifications can be used to post discussion comments by email.

To configure outbound email, use:

```
./docker_container_run.sh --email-host smtp.company.org --email-port 587 --email-user␣
↪... --email-pw ... --email-domain q.company.org
```

`--email-domain` sets the hostname used in the email address of outbound email. The other arguments set the SMTP relay server details.

Some of GovReady-Q's outbound emails can be replied to. When a user replies to a notification of a discussion comment, the reply's body is post as a new comment on the discussion. Currently we only support an incoming notification hook from Mailgun, and it is not yet configurable for the docker deployment. TODO

Container management and other options

Other options that can be passed on the command-line are:

Use `--name NAME` to specify an alternate name for the container. The default is `govready-q`.

Use `--relaunch` to remove an existing container of the same name before launching the new one, if an existing container of the same name exists. This simply runs `docker container rm -f NAME`.

Add `--debug` to start GovReady-Q in DEBUG mode, which enables nicer error messages. Do not use in production.

You can additionally pass parameters to the `docker container run` command by separating the Docker parameters from the GovReady-Q parameters with `--`, such as:

```
./docker_container_run.sh --address q.mydomain.com:80 -- -e VAR=VALUE
```

Adding and developing compliance apps

If you are using the Docker image to develop your own compliance apps, then you will need to bind-mount a directory on your (host) system as a directory within the container so that the container can see your app YAML files. To do so, start the container with the additional command-line argument:

```
--appsdevdir /path/to/apps
```

The directory may be empty but it must exist, and you must specify it as an absolute path (due to a Docker limitation).

The directory and its contents must also be readable — and writable, if you intend to use GovReady-Q's authoring tools — by the container process. The container process is running as user 1000/group 1000. Although the container is running as a user isolated from the host environment, filesystem permissions for mounted files are based on comparing the raw user/group IDs of the file's owner/group on the host to the raw user/group ID of the process running in the container. Consider granting user 1000 read/write permission to the files, plus execute (i.e. browse) permission to the directories, in the mounted path using ACLs:

```
setfacl -R -m u:1000:rwX /path/to/apps
```

Of course, do not do this if the host machine has a user 1000 that you do not trust.

If the directory is not empty, it should have subdirectories for each of your apps. For instance, you would have a YAML file at `/path/to/apps/my_app/app.yaml`.

To create your first app, you can run

```
docker container exec -it govready-q python3.6 manage.py compliance_app host your_new_
↪app_name
```

Replace `your_new_app_name` with an app identifier, which may contain letters, numbers, dashes, and underscores. `host` is always just `host` — don't change that.

If your new app does not appear in the compliance apps catalog, you may need to force the app catalog cache to be cleared by restarting the container:

```
docker container restart govready-q
```

Logs for Debugging

The container's console shows the output of container's start-up commands including database migrations and process startup. The container's console log can be accessed with

```
docker container logs govready-q
```

GovReady-Q application logs can be found in `/var/log` within the container to track status and assist with debugging. These files contain access logs and other program output.

- `/var/log/application-stderr.log` - GovReady-Q application standard error
- `/var/log/application-stdout.log` - GovReady-Q application standard out
- `/var/log/notificationemails-stderr.log` - GovReady-Q email notifications standard error
- `/var/log/notificationemails-stdout.log` - GovReady-Q email notifications standard error
- `/var/log/supervisord.log` - Supervisor daemon

Debugging "Internal Server Error" Messages

A special management command can be used to see the application log files to debug unhandled "Internal Server Error" (HTTP code 500) messages displayed in the browser to end users

```
docker container exec govready-q tail_logs
# Replace "govready-q" with name of your container or use container id
```

`tail_logs` takes the same arguments as Unix `tail`. For instance, add `-n 1000` to see the most recent 1,000 log lines, or add `-f` to continue to output the logs as the log files grow.

```
# most recent 200 lines of logs
docker container exec govready-q tail_logs -n 200

# real-time display of logs
docker container exec govready-q tail_logs -f
```

The log files can also be accessed by mounting `/var/log` with a Docker bind-mount or as a volume (and that's the only way to see the logs if `docker container exec` cannot be used in your environment).

Production deployment of the Docker container

The GovReady-Q container runs several processes, including an HTTP/application server and a background process for sending notification emails.

Secure deployments

The container's processes run exclusively as a non-root user with UID 1000 and GID 1000.

The container may be run with a read-only root filesystem (Docker's `--read-only` argument) so long as `/run`, `/tmp`, and `/var/log` are writable. When the `--dburl` argument is given to our `docker_container_run.sh` script, a read-only filesystem is activated using:

```
--read-only --tmpfs /run --tmpfs /tmp --tmpfs /var/log
```

The three directories can be made writable either by being mounted as tmpfs temporary filesystems, as above, or using a bind mount or a Docker volume. In production environments where the container is launched without our script, it is recommended to use tempfs for `/run` and `/tmp` and to mount `/var/log` to a volume.

Other management commands

See the uWSGI application server JSON process stats:

```
docker container exec govready-q uwsgi_stats
```

Updating to a new release of GovReady-Q

Periodically there will be a new release of GovReady-Q as an new image on the Docker Hub. Updating is easy by re-running the same commands again.

1) There may be an update to `docker_container_run.sh`. Since this script is not a part of the Docker image, you will need to get it again from this GitHub repository.
2) You should be using a persistent database as described above. When using a persistent database, it is safe to destroy the `govready-q` Docker container and start a new one to deploy an update.
3) Use the same arguments to `docker_container_run.sh` as when you started the container the last time, but add `--relaunch` to kill the previous container — you cannot have two containers with the same name or two containers listening on the same port. (You can change the name and port, as described above, if you would like to keep the old container running.)
4) When the new container starts, database migrations will be applied, if applicable.

For example:

```
# Update docker_container_run.sh, replacing the old script (with -O).
wget -O docker_container_run.sh \
    https://raw.githubusercontent.com/GovReady/govready-q/master/deployment/docker/
↪docker_container_run.sh
chmod +x docker_container_run.sh

# Remove old container and launch updated container.
./docker_container_run.sh --relaunch [your same command-line arguments]
```

Environment variables for launching the container without our run script

The following environment variables are used to configure the container when launching GovReady-Q using `docker run` or a container service (i.e., not when using our `docker_container_run.sh` helper script).

`HOST` - The domain name that GovReady-Q will be accessible at by end users. (Default: `localhost`)

`PORT` - The port that GovReady-Q will be accessed at by end users, typically either 80 (no HTTPS) or 443 (HTTPS). (Default: `8080`)

`HTTPS` - Set to `true` if GovReady-Q will be accessed by end users at an https: address. This must be done through a proxy that accepts HTTPS connections and passes the requests using HTTP to the Docker container. The proxy must set the `X-Forwarded-Proto: https` header. It is also permissible for the proxy to forward HTTP requests, and those requests will be automatically redirected to the https: URL. (Default: `false`)

`DEBUG` - Set to `true` to run in Django debug mode. (Default: `false`)

`DBURL` - Set to a database connection string as described in https://github.com/kennethreitz/dj-database-url. We recommend using PostgreSQL using a TLS server certificate, e.g. `postgresql://user:password@dbhost/govready_q?sslmode=verify-full&sslrootcert=/path/to/pgsql.crt` (although you'll have to figure out how to get the server certificate accessible via the container filesystem). (Default: Not set, which means using a Sqlite database stored in the container at `/usr/src/app/local/database.sqlite`, which will be ephemeral if the path is not mounted to the host or a Docker volume.)

`EMAIL_HOST`, `EMAIL_PORT`, `EMAIL_USER`, `EMAIL_PW`, and `EMAIL_DOMAIN` - For enabling outbound email. The host, port, username, and password settings specify a TLS-enabled SMTP server. `EMAIL_DOMAIN` is the domain name to use in outbound mail. (Default: Not set and outbound emails are dumped to logs for debugging.) To test the email configuration from the command-line, you can run `docker container exec -it govready-q python3.6 manage.py sendtestemail you@example.com`. If email is configured, you should not see any output and you should get a test email.

`FIRST_RUN` - If set to `1`, an administrator user will be created when the container launches and a randomly generated password will be given to the user and printed on the console, which will be visible in the container's logs. An organization named `main` will also be created.

`PROCESSES` - The number of concurrent requests that can be handled by the container. (Default: 4)

`SECRET_KEY` - The Django SECRET_KEY for session management. (Try this tool to generate one.)

`ADMINS` - The Django ADMINS setting, passed as raw JSON. Example: `[["Admin Name 1", "admin1@example.com"], ["Admin Name 2", "admin2@example.com"]]`. (Default: Empty list, i.e. `[]`.)

`SYSLOG` - The host and port of a syslog-compatible log message sink. (Default: None.)

`MAILGUN_API_KEY` - An API key for Mailgun which is used to validate incoming webhook requests from Mailgun when an incoming email is received, when Mailgun is configured to handle incoming mail. (Default: None)

`BRANDING` (downstream packaging only): You may override the templates and stylesheets that are used for GovReady-Q's branding by setting this environment variable to the name of an installed Django app Python module (i.e. created using `manage.py startapp`) that holds templates and static files. No such app is provided in the GovReady-Q published Docker image, so this variable can only be used by downstream image maintainers. See Applying Custom Organization Branding.

`PROXY_AUTHENTICATION_USER_HEADER` and `PROXY_AUTHENTICATION_EMAIL_HEADER`: GovReady-Q can be deployed behind a reverse proxy that authenticates users and passes the authenticated user's username and email address in HTTP headers. These environment variables correspond to the settings documented in Enterprise Login.

Running tests

GovReady-Q's unit tests can be run within the Docker container. After building the image:

```
docker container run --rm -it govready/govready-q:latest python3.6 manage.py test
```

Or once a Docker container running GovReady-Q is started (and named `govready-q`), use `exec` to begin a shell within the container, and then launch the unit tests:

```
docker container exec -it govready-q bash
python3.6 manage.py test guidedmodules
```

The functional tests run a headless Chromium web browser session. We welcome assistance figuring out how to get this to work in our Docker container. Chromium's process isolation capabilities seem to require special system privileges (i.e. `docker run --privileged --cap-add SYS_ADMIN`) or Chromium command-line flags (`--no-sandbox --disable-gpu`).

```
yum install -y chromium chromedriver
python3.6 manage.py test
...
selenium.common.exceptions.WebDriverException: Message: unknown error: Chrome failed␣
↪to start: exited abnormally
```

Populating sample data for manual testing and verification

If you wish to add sample data for testing purposes to your GovReady-Q image, run the following command (after `first_run` has completed):

```
docker container exec -it govready-q add_data --non-interactive
```

This will run a quickstart command to generate data in your GovReady-Q instance, as described in more detail in the Testing section of this documentation.

2.2.2 Deploying on macOS

Quickstart

macOS

Installing on macOS

GovReady-Q calls requires Python 3.6 or higher to run and several Unix packages to provide full functionality. Install the Homebrew package manager (https://brew.sh) to easily install Unix packages on macOS. Homebrew will install all packages in your userspace and not change native macOS Python or other libraries.

```
# install Homebrew package manager
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Now install Python3 and the required Unix packages.

```
# install dependencies using brew
brew install python3

# install other packages:
brew install unzip graphviz pandoc selenium-server-standalone magic libmagic
brew cask install wkhtmltopdf
```

Installing GovReady-Q

Clone GovReady-Q source code and install.

```
# clone GovReady-Q
git clone https://github.com/govready/govready-q
cd govready-q

# install Python 3 packages
pip3 install --user -r requirements.txt

# install Bootstrap and other vendor resources locally
./fetch-vendor-resources.sh
```

Run the final setup commands to initialize a local Sqlite3 database in *local/db.sqlite* to make sure everything is OK so far:

```
# run database migrations (sqlite lite database used by default)
python3 manage.py migrate

# load a few critical modules
python3 manage.py load_modules

# create superuser with initial account
python3 manage.py first_run
```

Start GovReady-Q

```
# run the server
python3 manage.py runserver
```

Visit your GovReady-Q site in your web browser at:

> http://localhost:8000/

Additional Details

We welcome assistance with installing GovReady-Q natively on MacOS.

2.2.3 Deploying on RHEL 7 / CentOS 7 / Amazon Linux 2

Quickstart

RHEL/CentOS 7

Installing on RHEL/CentOS 7

Instructions applicable RHEL 7, CentOS 7 and Amazon Linux 2.

GovReady-Q calls requires Python 3.6 or higher to run and several Linux packages to provide full functionality.

```
# if necessary, enable EPEL and IUS repositories
rpm -i https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm https://
↪rhel7.iuscommunity.org/ius-release.rpm

# install dependencies
sudo yum install \
unzip python36-pip python36-devel \
graphviz \
pandoc xorg-x11-server-Xvfb wkhtmltopdf \

# optional install gcc to build the uWSGI Python package.
sudo yum install gcc

# optional insall of postgress and/or mysql
sudo yum install postgresql mysql-devel
```

GovReady-Q calls out to *git* to fetch apps from git repositories, but that requires git version 2 or later because of the use of the GIT_SSH_COMMAND environment variable. RHEL stock git is version 1. Switch it to version 2+ by using the IUS package:

```
# if necessary, remove any git currently installed
yum remove git
# install git2u
yum install git2u
```

Installing GovReady-Q

Clone GovReady-Q source code and install.

```
# clone GovReady-Q
git clone https://github.com/govready/govready-q
cd govready-q

# install Python 3 packages
pip3 install --user -r requirements.txt

# install Bootstrap and other vendor resources locally
./fetch-vendor-resources.sh
```

Run the final setup commands to initialize a local Sqlite3 database in *local/db.sqlite* to make sure everything is OK so far:

```
# run database migrations (sqlite lite database used by default)
python3 manage.py migrate

# load a few critical modules
python3 manage.py load_modules

# create superuser with initial account
python3 manage.py first_run
```

Start GovReady-Q

```
# run the server
python3 manage.py runserver
```

Visit your GovReady-Q site in your web browser at:

> http://localhost:8000/

Additional Details

Deployment utilities

Sample `apache.conf`, `superviser.ini`, and `update.sh` files can be found in the source code directory `deployment/rhel`.

Creating a UNIX user named `govready-q`

You may find it useful to create a user specifically for GovReady-Q. Do this before installing GovReady-Q.

```
# Create user.
useradd govready-q -c "govready-q"

# Change permissions so that Apache can read static files.
chmod a+rx /home/govready-q
```

Upgrading pip

Upgrade `pip` because the RHEL package version is out of date (we need >=9.1 to properly process hashes in `requirements.txt`)

```
pip3 install --upgrade pip
```

Installing as UNIX user named `govready-q`

Switch to the govready-q user and install Q:

```
sudo su govready-q
cd
git clone https://github.com/govready/govready-q
cd govready-q
git checkout {choose the tag for the current released version}
pip3 install --user -r requirements.txt
./fetch-vendor-resources.sh
```

Installing drivers for Postgres, MySQL

```
# if you intend to use optional configurations, such as the MySQL adapter, you
# may need to run additional `pip3 install` commands, such as:
# pip3 install --user -r requirements_mysql.txt
```

2.2.4 Deploying on Ubuntu

Quickstart

Ubuntu 16.04

Installing on Ubuntu

Instructions provide basic guidance on setting up GovReady-Q on an Ubuntu 16.04 server with Nginx. These commands should be run from the root directory of the GovReady-Q code repository.

GovReady-Q calls requires Python 3.6 or higher to run and several Linux packages to provide full functionality.

```
# upgrade apt-get
apt-get update && apt-get upgrade -y

# install dependencies
apt-get install -y \
  unzip \
  python3 python-virtualenvpython3-pip \
  python3-yaml \
  nginx uwsgi-plugin-python3supervisor \
  memcached \
  graphviz

# optional install gcc to build the uWSGI Python package.
sudo yum install gcc

# optional insall of postgress and/or mysql
apt-get install -y postgresql mysql-devel
```

Installing GovReady-Q

Clone GovReady-Q source code and install.

```
# clone GovReady-Q
git clone https://github.com/govready/govready-q
cd govready-q

# install Python 3 packages
pip3 install --user -r requirements.txt

# install Bootstrap and other vendor resources locally
./fetch-vendor-resources.sh
```

Run the final setup commands to initialize a local Sqlite3 database in *local/db.sqlite* to make sure everything is OK so far:

```
# run database migrations (sqlite lite database used by default)
python3 manage.py migrate

# load a few critical modules
python3 manage.py load_modules

# create superuser with initial account
python3 manage.py first_run
```

Start GovReady-Q

```
# run the server
python3 manage.py runserver
```

Visit your GovReady-Q site in your web browser at:

> http://localhost:8000/

Additional Details

Deployment utilities

Sample `apache.conf`, `superviser.ini`, and `update.sh` files can be found in the source code directory `deployment/ubuntu`.

Creating a UNIX user named `govready-q`

You may find it useful to create a user specifically for GovReady-Q. Do this before installing GovReady-Q.

```
# create user
useradd govready-q -c "govready-q"

# change permissions so that Apache can read static files.
chmod a+rx /home/govready-q

# change to govready-q user
sudo su govready-q
```

Installing drivers for Postgres, MySQL

```
# if you intend to use optional configurations, such as the MySQL adapter, you
# may need to run additional `pip3 install` commands, such as:
# pip3 install --user -r requirements_mysql.txt
```

local/environment.json

Configure GovReady-Q by creating a file in `local/environment.json` with the following content:

```
{
  "debug": false,
  "admins": [["Name", "email@domain.com"], ...],
  "host": "q.<yourdomain>.com",
  "https": true,
  "secret-key": "something random here",
  "static": "/home/user/public_html"
}
```

You can use Django Secret Key Generator to make a secret-key value.

Prepare static files:

```
mkdir -p /home/user/public_html/static
python3 manage.py collectstatic --noinput
```

Set up supervisor to run the uwsgi daemon:

```
ln -sf `pwd`/deployment/ubuntu/supervisor.conf /etc/supervisor/conf.d/q.govready.com.
↪conf
service supervisor restart
```

2.2.5 Deploying on Windows (with Docker)

Quickstart

Windows

Installing on Windows (with Docker)

GovReady-Q can only be installed on Windows using Docker.

Make sure you first install Docker (https://docs.docker.com/docker-for-windows/install/).

Start

```
# Run the docker container in detached mode
docker container run --name govready-q --detach -p 8000:8000 govready/govready-q

# Create admin account and organization data
docker container exec -it govready-q first_run

# Stop, start container
docker container stop govready-q
docker container start govready-q

# View logs - useful if site does not appear
docker container logs govready-q

# To destroy the container and all user data entered into Q
docker container rm -f govready-q
```

Visit your GovReady-Q site in your web browser at:

http://localhost:8000/

Additional Details

We welcome assistance with installing GovReady-Q natively on Windows.

2.3 Deploying GovReady-Q in Production environments

These instructions assume that GovReady-Q is installed by the user `govready-q`, in the directory `/home/govready-q/govready-q/`.

To verify that this is the case, run the following command, and check whether GovReady-Q responds to HTTP requests (on `localhost:8000` by default).

```
cd /home/govready-q/govready-q/ && python3 manage.py runserver
```

If GovReady-Q is installed successfully, proceed with the rest of these configuration instructions. If it doesn't, see OS-specific install instructions.

2.3.1 Set basic configuration variables

Create a file named `local/environment.json` (ensure it is not world-readable) that contains site configuration in JSON, with some recommended settings:

```
{
  "debug": false,
  "host": "webserver.hostname.com",
  "https": true,
  "secret-key": "generate random string using e.g. https://www.miniwebtool.com/django-
↪secret-key-generator/",
  "static": "/home/govready-q/public_html/static"
}
```

Because of host header checking, to test the site again using `python3 manage.py runserver` you will need to visit it using `webserver.hostname.com` and not `localhost`. (Be sure to replace `webserver.hostname.com` with your hostname.)

Remember to Define Your `host`

The **DisallowedHost…Invalid HTTP_HOST header…You may need to add ':raw-html-m2r:'<your domain name>'' to ALLOWED_HOSTS** is a common error received when first trying to get GovReady-Q running on a server at a specific domain. The error indicates the domain you are trying to visit is not white listed in Django's special `ALLOWED_HOST` variable.

For security, Django requires white listing your server's domain(s) in the `ALLOWED_HOST` variable. Ordinarily this is hardcoded into the `settings.py` file. GovReady-Q allows the `ALLOWED_HOST` to be set by the `host` environment settings so the values can be passed at runtime.

- `host` must be defined, or GovReady-Q will default value to `localhost`

2.3.2 Setting up the Database Server

For production deployment, it is recommended to use dedicated database software, rather than SQLite.

The recommended database is PostgreSQL - see instructions on setting up Q with PostgreSQL

2.3.3 Setting up a Webserver

It's recommended to run a dedicated webserver software, such as Apache or Nginx, as a reverse proxy in front of the Q application (running through uWSGI). To read how to do this, see instructions on setting up Q with a reverse proxy webserver.

2.3.4 Creating the First User

Create the initial user and a "main" organization using:

```
python3 manage.py first_run
```

You should now be able to log into GovReady-Q using the user created in this section.

You can also use the Django admin to create organizations.

2.3.5 Other Configuration Settings

Set up email by adding to `local/environment.json`:

```
"admins": [["Your Name", "you@company.com"]],
"email": {
  "host": "smtp.server.com", "port": "587", "user": "...", "pw": "....",
  "domain": "webserver.hostname.com"
},
"mailgun_api_key": "...",
```

2.3.6 Updating Deployment

When there are changes to the GovReady-Q software, pull new sources and restart processes with:

```
# replace $DISTRO with an appropriate value.
# Currently-supported options include "rhel" and "ubuntu"
sudo -iu govready-q /home/govready-q/govready-q/deployment/$DISTRO/update.sh
```

As root, you can also restart just the Python/Django process:

```
sudo supervisorctl restart all
```

But this won't do a full update so don't normally do that (it won't restart the separate notifications process or generate static assets, etc.).

2.4 Deploying GovReady-Q for Development

This page provides additional tips for installing and run GovReady-Q in a mode suitable for making and testing changes to the software (i.e., in a Dev environment).

2.4.1 Quickstart

For local development, there is a quickstart script available to speed up environment setup. After installing system requirements through your package manager, run the following four commands in order to set up GovReady-Q in a new directory:

```
git clone https://github.com/govready/govready-q
cd govready-q
virtualenv -p python3 env
./quickstart.sh
```

This will set up a `local/environment.json` file suitable for a dev environment; set up local dependencies; and run the assorted initial manage.py commands (`migrate`, `load_modules`, `first_run`, etc.). Additionally, it can run common post-installation steps, based on user input.

The `quickstart.sh` script is set up to take user input, and is expected to be run interactively.

2.4.2 Creating local/environment.json file

When you first run GovReady-Q with `python manage.py runserver`, you'll be prompted to copy some JSON data into a file at `local/environment.json` like this:

```
{
  "debug": true,
  "host": "localhost:8000",
  "https": false,
  "secret-key": "...something here..."
}
```

This file is important for persisting login sessions, and you can provide other Q settings in this file.

2.4.3 Invitations on local systems

You will probably want to try the invite feature at some point. The debug server is configured to dump all outbound emails to the console. So if you "invite" others to join you within the application, you'll need to go to the console to get the invitation acceptance link.

2.4.4 Updating the source code

To update the source code from this repository you can `git pull`. You then may need to re-run some of the setup commands:

```
# if you set up virtualenv
git pull
source env/bin/activate
pip3 install -r requirements.txt
./fetch-vendor-resources.sh
python3 manage.py migrate
python3 manage.py load_modules

# if you did NOT set up virtualenv
git pull
pip3 install --user -r requirements.txt
```

(continues on next page)

(continued from previous page)

```
./fetch-vendor-resources.sh
python3 manage.py migrate
python3 manage.py load_modules
```

2.5 Setting up a Database for Production Workloads

The preferred production database for Q is PostgreSQL, but MySQL/MariaDB is also supported.

SQLite is supported in development environments, but not recommended for production use.

2.5.1 Setting Up Postgres

These instructions assume a separate database server and webapp server.

On the database server

On the database server, install Postgres. If using RHEL, CentOS, or similar:

```
yum install postgresql-server postgresql-contrib
postgresql-setup initdb
```

In `/var/lib/pgsql/data/postgresql.conf`, enable TLS connections by changing the `ssl` option to

```
ssl = on
```

and enable remote connections by binding to all interfaces:

```
listen_addresses = '*'
```

Enable remote connections to the database *only* from the webapp server and *only* encrypted with TLS by editing `/var/lib/pgsql/data/pg_hba.conf` and adding the line (replacing the hostname with the hostname of the Q webapp server):

```
hostssl all all webserver.hostname.com md5
```

Generate a self-signed certificate (replace `db.govready-q.internal` with the database server's hostname if possible):

```
openssl req -new -newkey rsa:2048 -days 365 -nodes -x509 -keyout /var/lib/pgsql/data/
↪server.key -out /var/lib/pgsql/data/server.crt -subj '/CN=db.govready-q.internal'
chmod 600 /var/lib/pgsql/data/server.{key,crt}
chown postgres.postgres /var/lib/pgsql/data/server.{key,crt}
```

Copy the certificate to the webapp server so that the webapp server can make trusted connections to the database server:

```
cat /var/lib/pgsql/data/server.crt
# place on webapp server at /home/govready-q/pgsql.crt
```

Then restart the database:

```
service postgresql restart
```

Then set up the user and database (both named `govready_q`):

```
sudo -iu postgres createuser -P govready_q
# paste a long random password

sudo -iu postgres createdb govready_q
```

Postgres's default permissions automatically grant users access to a database of the same name.

And if necessary, open the Postgres port:

```
firewall-cmd --zone=public --add-port=5432/tcp --permanent
firewall-cmd --reload
```

On the webapp server

On the webapp server, now check that secure connections can be made:

```
psql "postgresql://govready_q@dbserver.hostname.com/govready_q?sslmode=verify-full&
↪sslrootcert=/home/govready-q/pgsql.crt"
```

(It should fail if the TLS certificate file is not provided, if sslmode is set to `disable`, if a different user or database is given, or if the wrong password is given.)

Then in our GovReady-Q `local/environment.json` file, configure the database (replace `THEPASSWORDHERE`) by setting the following key:

```
"db": "postgresql://govready_q:THEPASSWORDHERE@dbserver.hostname.com/govready_q?
↪sslmode=verify-full&sslrootcert=/home/govready-q/pgsql.crt",
```

Then initialize the database content:

```
python3 manage.py migrate
python3 manage.py load_modules
```

And generate static files:

```
python3 manage.py collectstatic
```

2.6 Configuring a Reverse Proxy Webserver for Production Use

2.6.1 Setting Up Apache & uWSGI

Install Apache 2.x with SSL (back to being root):

```
yum install httpd mod_ssl
```

Copy the Apache config into place:

```
cp /home/govready-q/govready-q/deployment/rhel/apache.conf /etc/httpd/conf.d/govready-
↪q.conf
```

And then edit the file replacing `q.govready.com` and `*.govready.com` with your hostnames.

If you don't have a TLS certificate ready to use, create a self-signed certificate (replacing `webserver.hostname.com` with your hostname):

```
openssl req -new -newkey rsa:2048 -days 365 -nodes -x509 -keyout /home/govready-q/ssl_
↪certificate.key -out /home/govready-q/ssl_certificate.crt -subj '/CN=webserver.
↪hostname.com'
chmod 600 /home/govready-q/ssl_certificate.{key,crt}
chown apache.apache /home/govready-q/ssl_certificate.{key,crt}
```

If SELinux is enabled (`sestatus` shows `SELinux status:  enabled`), grant the Apache process access to these files as well as the site's static files:

```
chcon -v -R --type=httpd_sys_content_t /home/govready-q/govready-q/deployment/rhel/
↪apache.conf /home/govready-q/ssl_certificate.{key,crt} /home/govready-q/public_html
```

and grant Apache permission to make network connections so that it can connect to the Python/uwsgi backend running GovReady-Q:

```
setsebool httpd_can_network_connect true
```

Install `supervisor` which will keep the Python/Django process running and symlink our supervisor config into place:

```
yum install supervisor
ln -s /home/govready-q/govready-q/deployment/rhel/supervisor.ini /etc/supervisord.d/
↪govready-q.ini
```

Restart services:

```
service supervisord restart
service httpd restart
```

And if necessary open the web ports:

```
firewall-cmd --zone=public --add-port=80/tcp --permanent
firewall-cmd --zone=public --add-port=443/tcp --permanent
firewall-cmd --reload
```

GovReady-Q should now be running and accessible at your domain name. Follow the instructions in the main README.md for creating your first organization.

Setting up an HTTPS Certificate

The instructions above created a self-signed certificate to get the website up and running. To use Let's Encrypt to automatically provision a real certificate, install and run `certbot`:

```
yum install -y python-certbot-apache
certbot --apache -d webserver.hostname.com
# and follow the prompts
```

Then set it to automatically renew certificates as needed:

```
# edit root's crontab
crontab -e
```

(continues on next page)

(continued from previous page)

```

# insert at end:
30 2 * * * /usr/bin/certbot renew >> /var/log/le-renew.log
```

2.6.2 Setting up Nginx

Configure nginx to use nginx.conf in the govready-q directory:

```
# Turn off nginx's default website.
rm -f /etc/nginx/sites-enabled/default

# Put in our nginx site config.
ln -sf /home/govready-q/govready-q/deployment/ubuntu/nginx.conf /etc/nginx/sites-
↪enabled/yourdomain.com

service nginx restart
```

The nginx conf file assumes a certificate chain and private key are present at `/etc/ssl/local/ssl_certificate.crt/key`.

2.7 Multi-Container GovReady-Q and NGINX via Docker Compose

2.7.1 Overview

This directory contains configuration files that run two Docker containers, one for GovReady-Q and the other for NGINX, as a multi-container app. NGINX is used in a reverse proxy configuration, to handle incoming HTTP and HTTPS requests, which it then passes to GovReady-Q.

Use Docker Compose to manage the multi-container app.

Docker Compose commands are similar to, but different from, regular Docker commands. Read the Docker Compose docs for more details.

2.7.2 Set Up A Docker Host

Workstation

Install Docker and Docker Compose on your workstation.

- On Mac and Windows, Docker Compose is included as part of the Docker install.
- On Linux, after installing Docker, install Docker Compose. Also, you may want to grant non-root users access to run Docker containers.

Docker Machine

Docker Machine can be used to set up Docker host on either a local or cloud server. Once you have configured your shell to connect to a Docker host set up by Docker Machine, the Docker Compose commands you need to use will be the same as if you were using the Docker engine running on your workstation as the Docker host.

2.7.3 Get This Kit

Get the files by cloning the GovReady-Q repository.

```
git clone https://github.com/GovReady/govready-q.git
cd govready-q/deployment/docker-compose-nginx/
```

Make sure you are in the ‘‘docker-compose-nginx‘‘ directory.

Any `docker-compose` commands will need the `docker-compose.yml` file to know which containers to operate on.

2.7.4 SSL/TLS Certificates for HTTPS

There are self-signed certs including in the `nginx` directory. They are copied into the `nginx` container for `nginx` to refer to.

Self-signed certs are sufficient to allow GovReady-Q and NGINX to work together with your browser. However, you will get a security exception notice from your browser, and you will have to approve the “unsafe” exception to proceed.

To use real certs issued against a CA your browser will recognize, you can replace the `cert.pem` and `key.pem` files and issue the `docker-compose build` file, or you can mount a data volume with your certs in it to `/etc/pki/tls/certs/.`

Later versions of this project may include more documentation about the volume method, or other ways to include certs, such as Let’s Encrypt.

2.7.5 Build Images

You need to build images whenever you make changes to the Dockerfiles or `nginx` config file. If you don’t make changes, though, the command in the next section will automatically build the images.

To build the images:

```
docker-compose build
```

2.7.6 Run GovReady-Q + NGINX Multi-container App

To start the containers:

```
docker-compose up -d
```

Using the `-d` detaches the containers and runs them in the background.

If you prefer, you can omit `-d`, and then output will be printed to your console window. If you hit `^C`, the containers will shut down gracefully. If you hit `^C^C` they will be terminated immediately.

Two containers will be created, one for each “service” (as they’re called in Docker Compose).

Docker Compose gives these containers names like `docker-compose-nginx_govready-q_1` and `docker-compose-nginx_nginx_1`. These are three-part names, with the parts separated by underscores. `docker-compose-nginx` comes from the name of this project (the directory it’s in). The second element is the service name (`govready-q` or `nginx`). The third element is a serial number (ascending from 1) for multiple instances of the same service. The `docker-compose.yml` file here only specifies one instance, so the number will always be `1`.

You can check the status of the containers:

```
docker-compose ps
```

2.7.7 Specify Parameters

Before starting the containers, you can specify which GovReady-Q image to use, which database host to use, and the hostname of the Docker host. It's important to specify the correct hostname if you are using real TLS certs.

Set these environment variables (sample values provided, replace with your own values):

```
export GOVREADY_Q_HOST=ec2-nnn-nnn-nnn-nnn.us-east-1.compute.amazonaws.com
export GOVREADY_Q_DBURL=postgres://govready_q:my_private_password@grq-002.
↪cog63arfw9bib.us-east-1.rds.amazonaws.com/govready_q
export GOVREADY_Q_IMAGENAME=govready/govready-q-0.9.0
```

After setting the variables, continue with the "Run GovReady-Q + NGINX Multi-container App" section above.

If you don't set enviroment variables, these defaults are used:

```
export GOVREADY_Q_HOST=test.example.com
export GOVREADY_Q_DBURL=
export GOVREADY_Q_IMAGENAME=govready/govready-q
```

When no DBURL is specified, GovReady-Q uses an internal sqlite database.

2.7.8 Check Logs From A Container

Check the logs by specifying the service name:

```
docker-compose logs govready-q
```

```
docker-compose logs nginx
```

2.7.9 GovReady-Q Is Up

GovReady-Q will boot up, and be ready to answer web requests in 20-30 seconds.

It will answer HTTP on the standard port, 80, and HTTPS on the standard port, 443.

Visit https://localhost/. (Or `http://localhost`, which will be redirected to https by `nginx`.)

The default hostname used for this project is `test.example.com`. To check it, put this entry in your `/etc/hosts` file:

```
127.0.0.1       test.example.com
```

When you have `/etc/hosts` set up, visit https://test.example.com/

2.7.10 Execute A Script In A Container

You can exec a script inside one of the containers by specifying the service name. Unlike normal `docker`, you do *not* specify `-it` to make the exec interactive.

Here we are executing the `first_run` script inside the `govready-q` service/container.

```
docker-compose exec govready-q first_run
```

2.7.11 Stop And Remove Containers

To stop and remove containers:

```
docker-compose down
```

2.8 Environment Settings

2.8.1 Available Configuration Settings

The following environment variables are used to configure your GovReady-Q instance and can be configured through your local `local/environment.json` or passed in as environmental parameters when launching a container with GovReady-Q.

- `admins`: used to configure a display point of contact "Administrator" on site and unrelated to the configuration of actual administrators configured in the database.
- `branding`: used for custom branding
- `db`: if supplied, this is the DB connection used. See DB configuration.
- `debug`: should be `false` or absent in production environments. If set to `true`, turns on certain debug/development settings.
- *email*: used to configure access to a mail server for sending and receiving email. Ojbect has the following format: `{"host": "smtp.server.com", "port": "587", "user": "...", "pw": "....", "domain": "webserver.hostname.com"}`. See Configuring email and Other Configuration Settings. },.
- *govready_cms_api_auth*: used to store API key to interact with GovReady's CMS agent and dashboard. Not relevant to most users. See [GovReady-CMS-API](https://github.com/GovReady/GovReady-CMS-API).
- `host`: this is the domain name (HTTP `Host` header) used for root-level GovReady-Q pages. See also `organization-parent-domain`, which is used to construct organization subdomains (if using a different base domain).
- `https`: set to true to generate HTTPS headers and urls when site is running behind a proxy terminating HTTPS connections. See Configuring a Reverse Proxy Webserver for Production Use.
- `mailgun_api_key`: used to hold API key for using mailgun to send/receive emails.
- `memcached`: if setting is true, enable a memcached cache using the default host/port
- `organization-parent-domain`: this is the domain name (HTTP `Host` header) suffix used for organization-specific GovReady-Q pages.
- `organization-seen-anonymously`: show list of all created organizations to an anonymous (e.g., not signed-in) user on home page if set to true.
- `secret-key` - used to make instance more secure by contributing a salt value to generating various random strings and hashes. Do not share.
- `single-organization`: used to enforce single organization mode with "main" as subdomain of default organization instead of multi-tenant with different subdomains for different organizations.

- `static`: used to prepend a root path to the default `/static/` URL path for accessing static assets.
- *syslog*: used to set the host and port of a syslog-compatible log message sink. (Default: None.)
- `trust-user-authentication-headers`: used to activate reverse proxy authentication. See Proxy Authentication Server.

2.8.2 Production Deployment Environment Settings

A production system deployment may need to set more options in `local/environment.json`. Here are recommended settings:

```
{
  "debug": false,
  "admins": [["Name", "email@domain.com"], ...],
  "host": "q.<yourdomain>.com",
  "https": true,
  "secret-key": "something random here",
  "static": "/root/public_html"
}
```

2.8.3 Enterprise Single-Sign On Environment Settings

GovReady-Q supports Enterprise Login via IAM (Identity and Access Management). In this configuration, GovReady-Q is deployed behind a reverse proxy that authenticates users and passes the authenticated user's username and email address in HTTP headers.

To activate reverse proxy authentication, add the header names used by the proxy to your `local/environment.json`, e.g.:

```
{
    "trust-user-authentication-headers": {
      "username": "X-Authenticated-User-Username",
      "email": "X-Authenticated-User-Email"
    },
}
```

GovReady-Q must be run at a private address that cannot be accessed except through the proxy server. The proxy server must be configured to proxy to GovReady-Q's private address. See Enterprise Single-Sign On / Login for additional details.

2.8.4 Custom Branding Environment Settings

You may override the templates and stylesheets that are used for GovReady-Q's branding by adding a new key named `branding` that is the name of an installed Django app Python module (i.e. created using `manage.py startapp`) that holds templates and static files. See Applying Custom Organization Branding.

2.9 Enterprise Single-Sign On / Login

2.9.1 Proxy Authentication Server

GovReady-Q can be deployed behind a reverse proxy that authenticates users and passes the authenticated user's username and email address in HTTP headers. In this configuration:

- The user points their browser to the reverse proxy authentication server.
- The proxy authenticates users and proxies the request to GovReady-Q if and only if the user is authenticated and authorized to access GovReady-Q. The proxy passes the user's username and email address in HTTP headers of the proxy's choosing.
- GovReady-Q will create a user account for a new user or treat the user as logged in as soon as the user requests a page. Therefore, there is no sign-up or log-in process within GovReady-Q when a proxy authentication server is used.
- All other authentication methods to GovReady-Q are disabled when proxy authentication is enabled. Therefore you should ensure that the Django admin's username matches the admin's username as provided by the proxy server. Otherwise, you will lose access to the admin page. However, if there is a mismatch, you may disable proxy authentication, log in to the Django admin with your admin username and password, and change your admin username to match the username sent by the proxy server.
- GovReady-Q must be run at a private address that cannot be accessed except through the proxy server.

To activate reverse proxy authentication, add the header names used by the proxy to your `local/environment.json`, e.g.:

```
{
  "trust-user-authentication-headers": {
    "username": "X-Authenticated-User-Username",
    "email": "X-Authenticated-User-Email"
  },
}
```

The proxy server must be configured to proxy to GovReady-Q's private address. But the `host` and `https` settings in GovReady-Q's `local/environment.json` file must reflect the host and protocol used in the URL the *end user* uses to access GovReady-Q. They do *not* need to match the address that the proxy server uses to reach the GovReady-Q server.

Per the Django Documentation on authentication using REMOTE_USER, you must be sure that your proxy server always sets or strips the special username and email headers, including headers that normalize to the same Django key (in particular headers with underscores), from the client request and **does not permit an end-user to submit a fake (or "spoofed") header value**.

We have an example reverse proxy authentication server at https://github.com/GovReady/govready-q/tree/master/tools/simple_iam_proxy_server which can be used for debugging purposes.

2.10 Applying Custom Organization Branding

The look and feel of GovReady-Q can be customized a bit by overriding the Django templates that are used to construct the site's pages and by serving additional static assets.

Custom branding can contain static assets (such as a logo image) and HTML template overrides. Branding is packaged into a directory with a particular directory layout and some Python boilerplate code that allows GovReady-Q to find the branding files. The directory is placed inside the main GovReady-Q directory, and an application setting is used to activate it.

Before setting out to create custom branding, make sure you have GovReady-Q set up for development on your workstation. You'll need a working setup of GovReady-Q to create the branding directory and to test your changes.

2.10.1 Creating the branding directory

Custom branding is packaged inside what Django confusingly calls an application, but it is just a packaged sub-component of a website. To create a new branding package directory, change to the directory where you have GovReady-Q set up. Then run:

```
python3 manage.py startapp sample_branding
```

This command creates a new directory called `sample_branding` with Python boilerplate code to make it a valid Django "application."

Make directories for storing the custom static assets and templates:

```
mkdir sample_branding/static
mkdir sample_branding/templates
```

2.10.2 Activate the branding package

Next, let your development installation of GovReady-Q know that you want to use the custom branding package. In your `local/environment.json` file, add a setting named `branding` and set it to the name of the custom branding package directory.

```
"branding": "sample_branding",
```

See Environment Settings for more information about the `local/environment.json` file. Note that for the file to be valid JSON the last setting cannot have a trailing comma.

2.10.3 Overriding templates

Any of the templates that make up GovReady-Q's frontend can be overridden. The full list of templates can be browsed in GovReady-Q's GitHub repository at

https://github.com/GovReady/govready-q/tree/master/templates

Start by trying to override the `navbar.html` template, which is inserted at the top of every page. Use your favorite text editor to create a file at `sample_branding/templates/navbar.html`. Copy the content of GovReady-Q's stock `navbar.html` from https://github.com/GovReady/govready-q/blob/master/templates/navbar.html into it. (GitHub's "Raw" button is handy for getting a clean version to save or copy/paste.)

At the bottom of the file, add some custom HTML, such as:

```
<div>
  <b>Welcome to my organization&rsquo;s custom site!</b>
</div>
```

Start GovReady-Q on your workstation (see the development docs) and visit a page. You should see your new content below the navbar at the top of every page.

2.10.4 Adding custom CSS

You can also add a custom CSS stylesheet to your branded GovReady-Q by taking the following steps:

a) Add the CSS file as a static asset.

b) Insert a `<link rel="stylesheet" href="...">` tag into the `<head>` section of each page's HTML by overriding the `head.html` template.

To create the static asset, make a new file named `sample_branding/static/custom.css`. Let's say you want to make the background color of each page red. The file should contain:

```
body {
    background: red !important;
}
```

Then override the `head.html` template. GovReady-Q's base for `head.html` is empty — its purpose is only to allow you to add to the `<head>` element. So create a new file at `sample_branding/templates/head.html` and put in it:

```
{% load static %}
<link rel="stylesheet" href="{% static "custom.css" %}">
```

See the Django documentation for static files for more information about the `static` template tag.

Open any page in your locally running GovReady-Q and you should see that the background color of every page has changed.

2.10.5 Keeping your templates up to date

With each new released version of GovReady-Q, there is the possibility that the stock templates have changed. Some changes may require you to re-engineer your template overrides to preserve functionality.

2.10.6 Creating a custom Docker image

If your organization is deploying GovReady-Q using Docker, you will need to embed your custom branding package within a Docker image. You have two options:

1. Modify GovReady-Q's stock Dockerfile, i.e. the one in GovReady-Q's source code, to add and activate your branding package and then *build your own GovReady-Q Docker image* from the GovReady-Q source files that you cloned from GitHub.
2. Make your own Dockerfile that *uses a released GovReady-Q image as its parent image* and adds to it just the steps needed to add and activate your branding package.

Creating your own Dockerfile that uses a released GovReady-Q image as its parent image

We recommend method 2. To create your own Dockerfile that uses a released GovReady-Q image as its parent image, create a new `Dockerfile` in your branding package directory, e.g. a new file named `Dockerfile` in the `sample_branding` directory you created earlier.

Then choose which parent image you will use from the available GovReady-Q tags. Each tag corresponds to a release version. Your Dockerfile begins with a `FROM` line that combines `govready/govready-q:` with the tag name you choose. In this example we use the `latest` tag which is an alias for the most recent version of GovReady-Q:

```
FROM govready/govready-q:latest
```

The subsequent commands in your Dockerfile configures the container, picking up where the parent image's Dockerfile leaves off. For more information about the parent image, refer to GovReady-Q's Dockerfile on GitHub.

Your Dockerfile's next step is to add your branding package into the image in a directory named `branding`:

```
RUN mkdir branding
COPY . branding
```

Finally, you'll need some commands to adjust permissions, to activate the branding package when GovReady-Q starts, and to prepare the static assets to be served. The complete Dockerfile should look like this:

```
# Build an image on top of the stock GovReady-Q image.
FROM govready/govready-q:latest

# The parent Dockerfile ends with 'USER application' to run the
# container as a non-privileged user. But we need to go back to
# root to add additional files and then switch back to the non-
# root user at the end.
USER root

# Copy our public app files into place.
RUN mkdir branding
COPY . branding

# Activate the branding package. The environment variable is read
# by dockerfile_exec.sh in the GovReady-Q parent image. And modifying
# /tmp/environment.json is necessary at this step so that collectstatic
# picks it up below.
ENV BRANDING branding
RUN sed -i "s/}/,\"branding\": \"branding\" }/" /tmp/environment.json

# Flatten static files. The base image did it once, but we may have
# added new static files so we must do it again.
RUN python3.6 manage.py collectstatic --noinput

# Run the container's process zero as this user --- see above.
USER application

# Check that everything looks good.
RUN python3.6 manage.py check
```

Finally you can build and test your custom image.

Building your docker image

If you were in the GovReady-Q sources directory, move into your branding package directory:

```
cd sample_branding
```

Then fetch the parent image and build your image:

```
docker image pull govready/govready-q:latest
docker image build --tag myorg/govready-q-branded:latest .
```

(Substitute the right tag depending on the tag you chose for the `FROM` line in your Dockerfile.)

Test that your image works by launching a new container based on your image:

```
docker container run --rm -it -p 127.0.0.1:8000:8000 myorg/govready-q-branded:latest
```

Once GovReady-Q is running in the container, visit it at `http://localhost:8000`. Use CTRL+C in the console to terminate and destroy the test container running your image.

For more about running GovReady-Q with Docker, see Deploying with Docker.

CHAPTER

THREE

PERMISSIONS

This document describes the user permissions model in Q.

3.1 What Q tracks

GovReady-Q tracks the following major entities

- Users - individuals with logins to an installed instance of Q
- Organizations - entities, e.g., companies, around with data in Q is segmented
- Folders - collections of Projects
- Projects - instantiations of Compliance Apps
- Membership - associating individual users with organizations and systems
- Tasks/Modules - coherent grouping of questions and educational content
- Questions/Answers - specific snippet of content within a Task
- Templates - drives automatic generation of artifacts, supporting variable substitution

3.2 Users

3.2.1 Global user data

A User is authenticated by a unique username and a hashed password.

Each User has one or more email addresses associated with their account and notification email settings.

User accounts — i.e. the fields above — are global to a Q deployment. They are *not* segmented by Organization. For instance, if a User has been created for one Organization, they sign in — and not sign up — into other Organizations. Changing a User's password changes it for all Organizations. (Of course, a User will not be *authorized* to access all Organizations. See the next section on Organizations.)

3.2.2 Segmented user data

Each User additionally has a unique "organization user profile" associated with each organization to support the same User having different roles at different organizations. (This user profile is an "account project" that holds additional User information including a User's full name, profile photo, etc. In short, each User has a different profile in each Organization.

The profile information can be seen by all other members of the Organization because it is used in the history of question answers, notifications, discussions, and many other places.

(The User is the only "member" of their account projects (see Project *membership* below), which means they are the only User who can edit the information.)

3.2.3 System staff

Users marked as `staff` in the Django admin can see Q Analytics.

3.3 Portfolios

Portfolios are used to segment Projects and the data contained within them to create data isolation at a logical level.

3.3.1 Access to Portfolios

Each Portfolio begins with an 'owner' creating the portfolio. When any user signs up, a portfolio is automatically created with the title as the users username and assigning ownership to the user. An owner of a portfolio can grant other users access to their porfolio. If a user has access to a portfolio, they will have access to the projects within that portfolio. If an owner of a portfolio removes a users access to a portfolio, they will lose access to the projects within that portfolio.

3.4 Folders

A Folder is a collection of one or more Projects (see below) within the same Organization.

Folder permissions are based in part on Project permissions:

- A user can see that a Project is a part of a Folder if the user has *read* access on the Project.
- A user can add Projects to a Folder or rename a Folder if the user is an *administrator* of any Project within the Folder *or* is an *administrator* of the Folder itself. These users may not be able to see all Projects within the folder if they do not have *read* access to those Projects, but they will be told how many Projects they can't see in the Folder.

There is no separate "read" permission on a Folder. A Folder can be seen just when a user has *read* access on a Project within it or is an *administrator* of the Folder itself.

3.5 Projects

Each time an app is started from the Compliance Store, a new Project is created. A Project represents the instantiated app and is comprised of a collection of Tasks. Every Project belongs to exactly one Organization.

3.5.1 Membership

Projects have zero or more Users who are *members* and zero or more Users who are *administrators*.

3.5.2 Read Access

Any access to a Project requires *read* access, which is granted if any of the following are true:

- They are a *member* or *administrator* of the Project.
- They have *read* access to any Task in the project.
- They are a guest participant in a Discussion within the Project.

(This is a subset of the requirements for membership in an Organization, therefore *read* access to a Project guarantees membership in the Organization it belongs to.)

3.5.3 Operations

Project *members* can begin Tasks listed on Project pages, either by adding apps from the Compliance Store or starting Tasks for modules contained in the Project's app. Project *members* can also invite guests to discussions.

Only *administrators* can send invitations to add new project members, import and export Project data, and delete Projects.

The Tasks (the questions and answers) within a Project are further restricted (see Tasks below).

3.5.4 New Projects

Any member of an Organization can create a new Project within that Organization by starting a new Compliance App and becomes the Project's first administrator.

When a User creates a new Project, they are offered Compliance Apps from AppSources whose `Available to all` option is checked and from AppSources that don't have 'Available to all' checked but explicitly list the Organization in its 'Available to orgs' list.

New Projects are added into a new or existing Folder (for existing folders, see Folder permissions above).

3.6 Tasks

A Task is a set of questions and answers. Tasks represent the state of a Project — each Project has a root Task — as well as the state of all the modules started within the Project.

Each Task belongs to exactly one Project. Each Project has exactly one root Task.

A Task has an editor, which is the User who has primary responsibility for completing the Task.

A User has both *read* and *write* access to a Task if any of the following are true:

- They are the editor of the Task.
- They are a *member* or *administrator* of the Project that the Task belongs to.

A User with *read* access can see the Task on the page for the Project that it belongs to and can see all of its questions, answers, and outputs and can start a Discussion on questions.

A user can also see a particular question within a Task (and its answers and some Task metadata, but not other questions or Task outputs) if they are a guest in a Discussion on that question.

A User with *write* access to a Task can answer questions within the Task (which sometimes involves starting new Tasks which they become the editor of), invite other users to become the Task's new editor, and delete/undelete the Task (although there is no UI for that currently).

CHAPTER

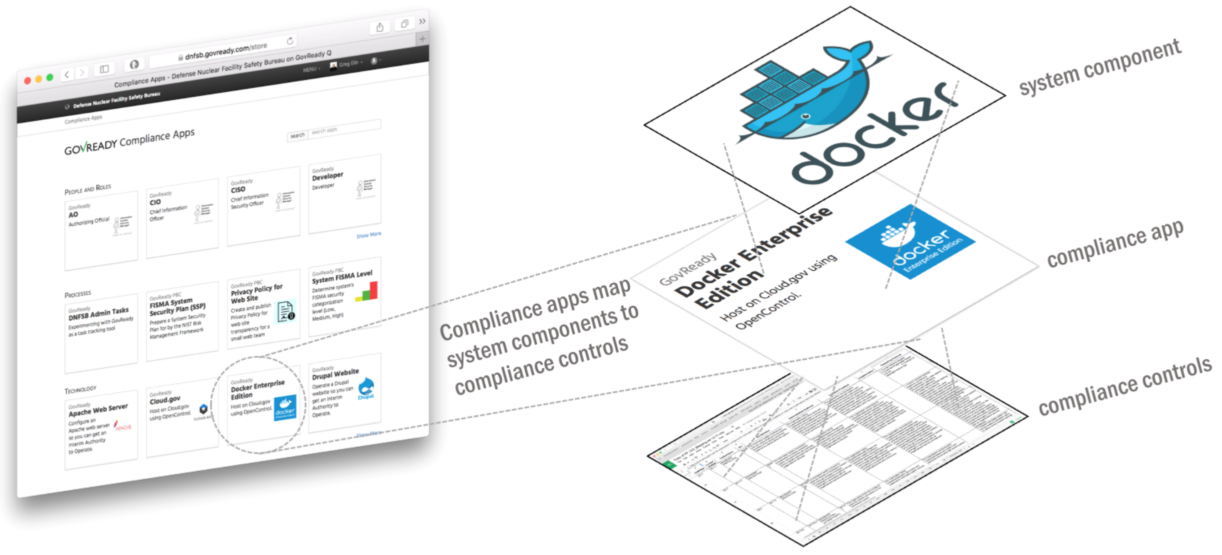
FOUR

AUTHORING COMPLIANCE APPS

4.1 Understanding Compliance Apps

Compliance apps map IT System components to compliance controls. A "component" can be any part of a system that contributes to its operation including organizational processes.

Compliance apps collect and assess information about one or more system components and translate that information to compliance documentation.



Compliance apps can collect information about a system component from people (via web-based questionnaires) and from system components (via an Automation API).

4.1.1 Compliance Apps are Collections of Modules

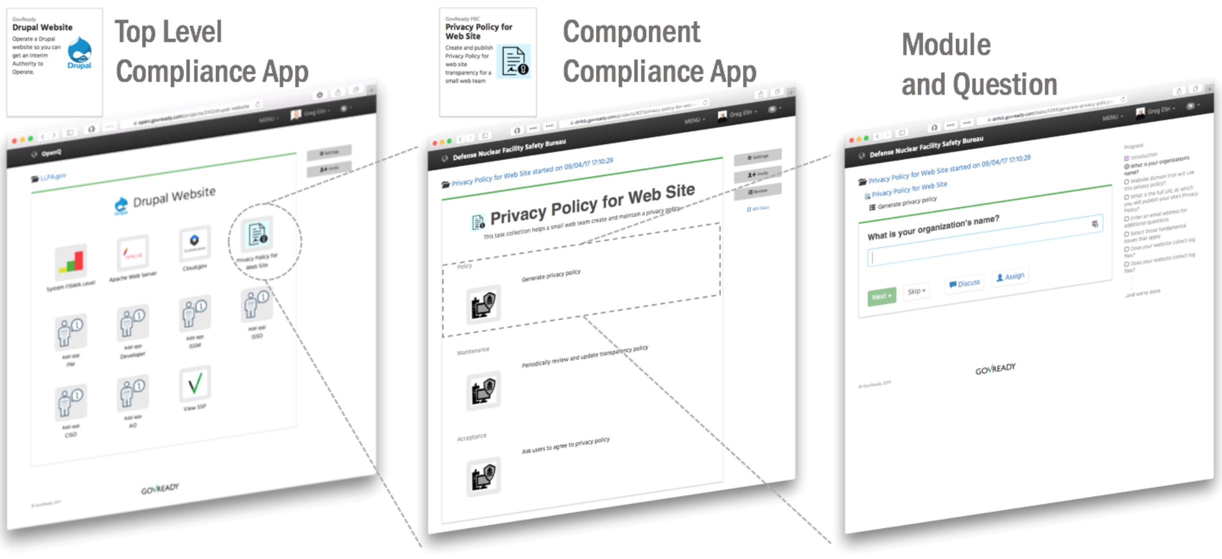
A Compliance app is a collection of "modules" for gathering information. A module is a collection of questions and output documents. A module can have just questions and output documents, just output documents and no questions, or both questions and output documents.

An "app" is a collection of "modules," one of which must be named "app." Modules are linear sequence of questions presented to users that produces zero or more output documents. Modules are stored in YAML files. Output documents of various types are supported such as markdown, HTML, and YAML. (See Modules, Questions, and Documents for documentation on writing modules.)

The typical user experience will be to first pick a “top level” app from the compliance catalog representative of IT System, then pick the “component” apps that represent the specific components of the IT System, and then iteratively complete the questions within the component apps modules.

Technically speaking, a top level app is a module containing questions whose answers are other apps.

The below diagram depicts an exploded view of the relationships between a top level app to a component app to modules and questions.



4.1.2 App Structure

Each app is defined by a set of YAML files and asset files stored in the following directory structure:

```
app_name
├── README.md
├── app.yaml
├── assets
│   ├── app.png
│   ├── image_one.yaml
│   ├── image_two.yaml
│   └── ...
├── module_one.yaml
├── module_two.yaml
└── ...
```

By convention, each app is required to have `app.yaml` file which holds metadata for displaying the app in the compliance apps catalog, such as its title and description, and an `assets/app.png` graphic which displays as the app’s icon. `app.yaml` also holds the top-level module questions which define the layout of the app’s main screen once it is started by the user. The contents of `README.md` are also displayed in the apps catalog.

Other module YAML file may be includes in the app as well, as needed.

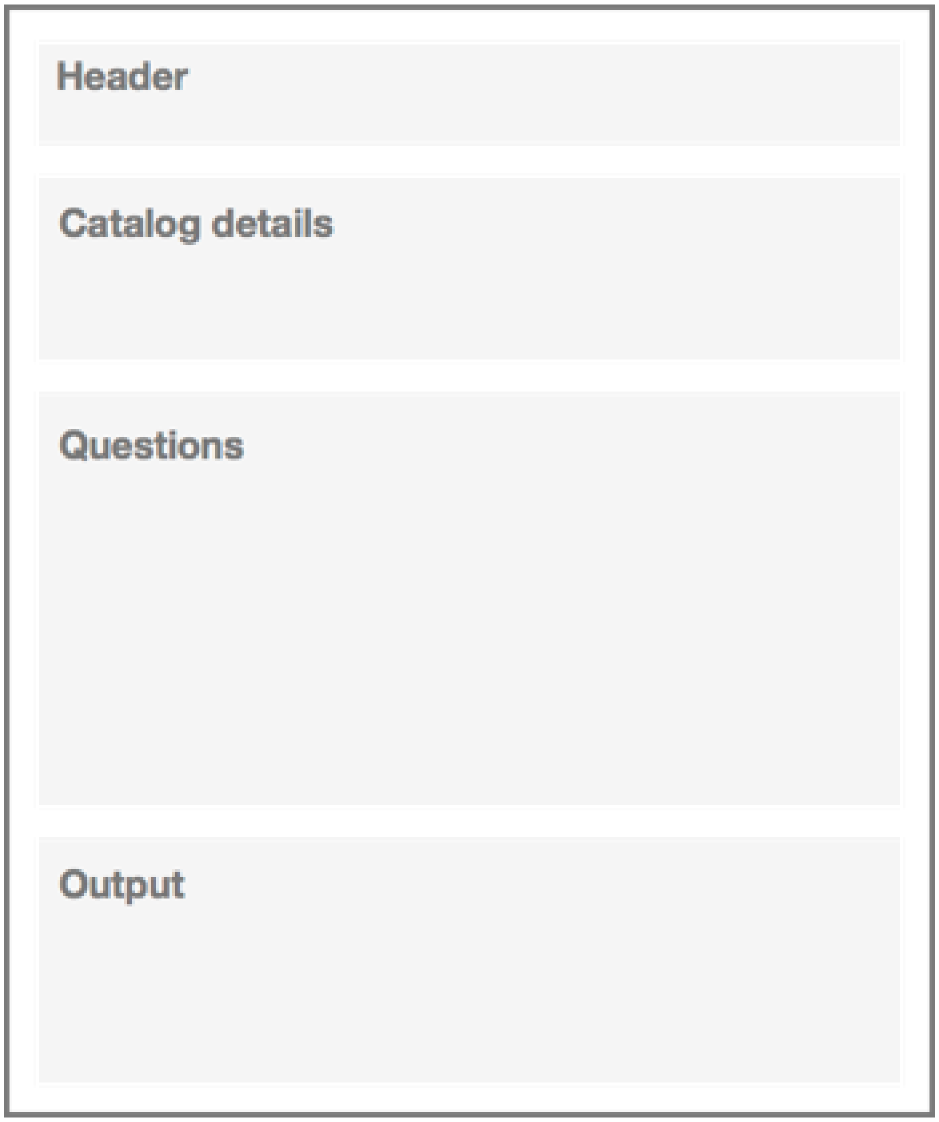
The `assets` subdirectory can contain any static assets that will be served when showing the app’s modules for the user, such as images included in document templates. A file typically named `app.png` in the assets directory is the app’s icon, which is displayed when browsing the app catalog as well as when the app is used within another app, if `icon: app.png` is specified in `app.yaml`.

4.1.3 App YAML

The `app.yaml` file that exists in every app serves two purposes:

1. It includes app catalog information, i.e. metadata, that will be shown in the app directory, such as the app's short and long description, version number, vendor, etc.
2. It also defines a module (see Modules, Questions, and Documents) which defines the top-level layout of the app. The module may only contain questions whose type are `module` or `module-set`.

The `app.yaml` file looks like this:



```
id: app
title: My App
type: project
icon: app.png # refers to file in app's assets directory
protocol: # for inner apps only
- globally_unique_protocol_name

catalog:
```

(continues on next page)

(continued from previous page)

```
  categories:
    - Category Name
    - Another category name
  vendor: GovReady PBC
  vendor_url: https://www.govready.com
  status: Operational
  version: 0.6
  version-name: First Release
  source_url: https://github.com/GovReady/govready-app-example
  description:
    short: |
      One-line description of the app here, using Markdown.
    long: |
      Long description of the app here only if README.md is
      not present.

      It can be multiple paragraphs and is Markdown.
  recommended_for:
    - key_short: Org
      value: Medium
    - key_short: Tech
      value: Drupal
    - key_short: Role
      value: Dev

questions:
  - id: item1
    title: Do A Thing
    type: module
    module-id: module1 # refers to module1.yaml within this app
    tab: TabName
    group: GroupName
  ... more questions here ...

output:
  - tab: TabName
    format: markdown
    template: |
      This (optional) content will appear at the top of the TabName tab.
```

The questions in the app YAML file can only be of type `module` and `module-set`. The questions can specify a `module-id` to refer to another module within the same app or a `protocol` to allow the user to choose any app that has a matching `protocol` value set at the top level of the YAML file. See Modules, Questions, and Documents for details on these question types.

A `module` YAML structure is identical to `app.yaml` structure but without the catalog details section.

Hide "Question Skip" Buttons

As of version 0.8.6, the "I don't know" and "It doesn't apply" buttons to skip questions can be hidden from users.

We recommend you never use this feature.

This feature was added to support the use case of complex, legacy questionnaire assessments that (1) cannot be changed easily and (2) provide a better user experience when users get stuck and start a discussion instead of later learning that skipping a question caused them to miss many other questions.

Some background is useful. GovReady-Q was designed for users to *love* easily and quickly answering as few questions as possible to *generate* the information that organizational processes need. Users love the option to skip questions. Skipped questions empower users to move fast, answer what they can, and iteratively complete work. That gets information to you quickly and reduces cycle time. Skipped questions also provide instant feedback that a user doesn't have information readily available. Frequently skipped questions indicate stumped users, and the need to rethink the question or use multiple questions and interstitials to better guide users.

We've all experienced the frustration of not understanding what is being asked of us, not knowing whom to ask for help, or knowing the question asked does not apply. This frustration turns galling and Kafkaesque when the party demanding compliance confounds our ability to comply.

Fight-or-flight response kicks in when users feel trapped. Users start to avoid your process or combat it. They can (unfairly) transfer their frustration onto your role, or worse, onto you. Instead of helping to pull your colleagues into your process, they feel you are pushing them away.

So if you really, *really* need to hide the skip buttons to make the experience easier for your users, add a `hidden-buttons` array key to the `app.yaml` file and list the skip buttons to hide. The example snippet below hides just the "I don't know button".

```
id: app
title: My App
type: project
icon: app.png # refers to file in app's assets directory
protocol: # for inner apps only
- globally_unique_protocol_name
hidden-buttons:
- no-idea
```

The possible YAML array values for the buttons are `no-idea` ("I have no idea"), `not-applicable` ("It doesn't apply"), `not-now-button` ("I'll come back"), and `not-sure-button` ("Unsure").

NOTE: As of version 0.8.6, the "I'll come back" and "Unsure" functionality been removed from the UI for all cases because of poor user experience but are preserved for legacy data and potential future use with a better UI.

4.1.4 Top Level Apps

Apps that describe the required components of a compliant IT system are considered "Top Level" apps. Each question in a Top Level app specifies a type of compliance app (e.g., a compliance app "protocol") that is needed to represent that component.

4.1.5 Adding Apps to GovReady-Q Deployments

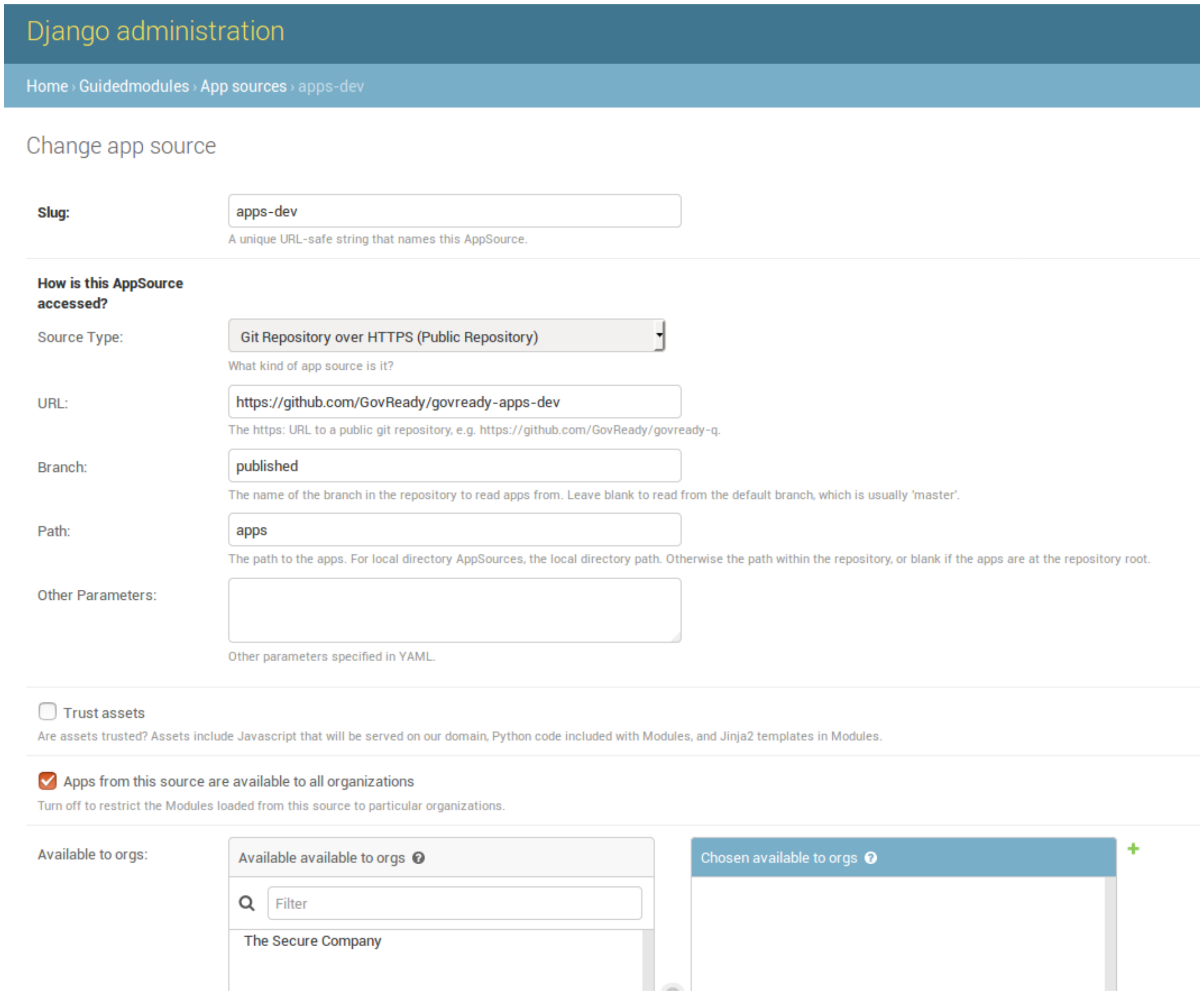
Separating compliance apps from the compliance server enables a much richer ecosystem and virtuous cycle of innovation than having everything embedded exclusively within the compliance server. A GovReady-Q deployment can pull app and module content from local directories and git repositories. An organization using GovReady-Q can freely mix compliance apps from third parties with private compliance apps located only on their network.

Compliance apps are very much like modular plugins that customize the compliance server to the unique system and components of the organization.

This leaves the need to specify which compliance apps are available to a compliance server deployment. This specification of available apps is known as an "app source" and is done with a JSON "spec" file entered in the `AppSource` model via the Django admin interface.

The process is currently a bit clumsy with terminology that reflects the software's evolution toward the app concept. Nevertheless, the approach provides flexibility of sourcing apps from local file systems and public and private git repositories. And each source specifies a virtual filesystem from which one or more top level apps and compliance apps can be found located.

The below screenshot of the AppSource module in the Django admin interfaces shows the JSON "spec" file.



Django administration

Home › Guidedmodules › App sources › apps-dev

Change app source

Slug: apps-dev
A unique URL-safe string that names this AppSource.

How is this AppSource accessed?

Source Type: Git Repository over HTTPS (Public Repository)
What kind of app source is it?

URL: https://github.com/GovReady/govready-apps-dev
The https: URL to a public git repository, e.g. https://github.com/GovReady/govready-q.

Branch: published
The name of the branch in the repository to read apps from. Leave blank to read from the default branch, which is usually 'master'.

Path: apps
The path to the apps. For local directory AppSources, the local directory path. Otherwise the path within the repository, or blank if the apps are at the repository root.

Other Parameters:
Other parameters specified in YAML.

☐ Trust assets
Are assets trusted? Assets include Javascript that will be served on our domain, Python code included with Modules, and Jinja2 templates in Modules.

☑ Apps from this source are available to all organizations
Turn off to restrict the Modules loaded from this source to particular organizations.

Available to orgs:

The AppSource module also contains fields to indicate to which subdomains of the deployment the source's apps are available.

App Source virtual filesystem layout

Whether the source is a local directory or a git repository, the source must have a directory layout in which each app is stored in its own directory. (The directory name becomes an internal name for the app.) For instance:

```
app1/app.yaml
app1/...other_app1_files
app2/app.yaml
app2/...other_app2_files
...
```

Updating modules

After making changes to modules or AppSources for system modules (like account settings), run `python3 manage.py load_modules` to pull the modules from the sources into the database. This only updates system modules.

Other modules that have already been started as apps will not be updated. Each time you make a change to an app, you can reload changes using the app authoring tool in GovReady-Q.

4.2 Compliance App Authoring Tutorial

This is a step-by-step guide to creating compliance apps using the Docker version of the GovReady-Q Compliance Server.

In this guide you will learn how to:

- Start and configure the Docker version of GovReady-Q
- Create a compliance app
- Edit a compliance app's YAML files
- Edit a compliance app using GovReady-Q's authoring tools
- Deploy the app to a production instance of GovReady-Q and storing apps in a source code version control repository

4.2.1 Step 1: Prepare your local environment

Create a folder on your workstation

GovReady-Q compliance apps are generally developed in an off-line development environment, usually on the app developer's macOS or Linux workstation — any environment that can run Docker. In this environment, the compliance app data files will be stored in a local directory. This guide assumes the use of a local workstation for development and discusses production deployment at the end.

(Once the apps are ready to be published to the rest of the organization, the apps can be uploaded to a git repository, such as GitHub or an on-premise equivalent. The production instance of GovReady-Q will typically read compliance apps from the git repository directly and not from a local disk.)

On the development workstation, create a folder to hold GovReady's install script, the GovReady-Q database (in development, Sqlite is used), and the compliance apps that you will be authoring. The folder can be anywhere:

```
mkdir /path/to/dev_directory
cd /path/to/dev_directory
```

Install Docker

If you haven't already done so install Docker on the workstation and, if appropriate, grant non-root users access to run Docker containers (or else use `sudo` when invoking Docker below).

4.2.2 Step 2: Install the GovReady-Q Compliance Server, Docker version

Starting the Docker container

Next download GovReady's docker_container_run.sh script. This script simplifies passing various settings to create and configure the `govready-q` docker container that we will use for local development.

```
wget https://raw.githubusercontent.com/GovReady/govready-q/master/deployment/docker/
↪docker_container_run.sh
chmod +x docker_container_run.sh
```

`docker_container_run.sh` supports a variety of advanced configuration settings via command line parameters. The ones we care about for developing compliance apps are:

- `--sqlitedb /path/to/govready-q-database.sqlite`, which sets an absolute path to a Sqlite database that holds all persistent information across container runs
- `--appsdevdir /path/to/apps`, which sets an absolute path to the directory in which app YAML files will be developed
- `--relaunch`, which removes any existing `govready-q` Docker container if one is running

Download and start GovReady-Q:

```
./docker_container_run.sh --sqlitedb `pwd`/database.sqlite --appsdevdir `pwd`/apps --
↪relaunch
```

Note that `pwd` is used to ensure the paths are absolute.

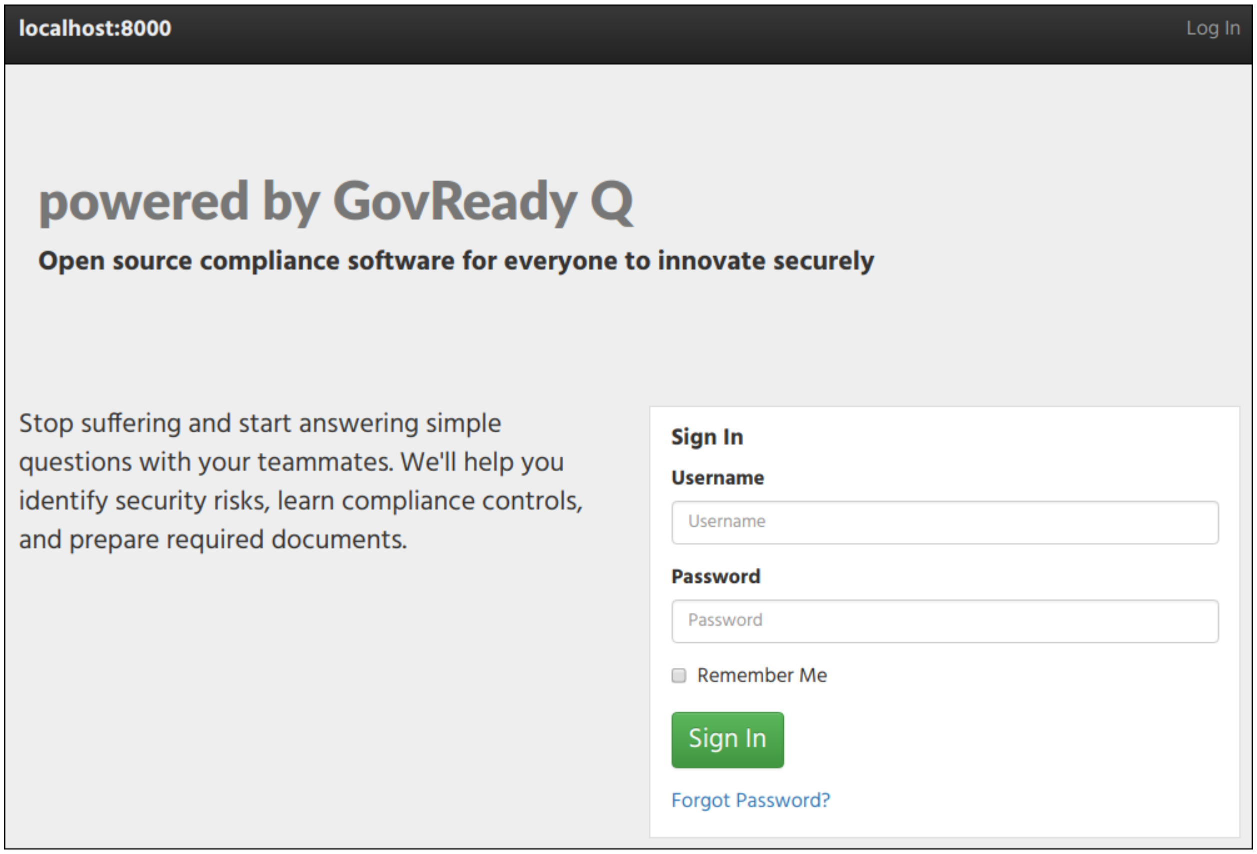
The script will download the govready/govready-q image from the Docker Hub, which could take a few minutes. It will then start a new Docker container named `govready-q` and will launch the Q source code within it.

When the container is launched it will let you know the URL to visit:

```
GovReady-Q has been started!
Container Name: govready-q
Container ID: d99e8ac2d6a761cfd7be7f94bd01d5f7115efd66714064f7b1f0f6c09b74c269
URL: http://localhost:8000
```

(You can change the hostname and port by adding e.g. `--address q.company.com:8010`.)

It takes about 15 seconds for the GovReady-Q server to be ready. Open the URL (e.g. http://localhost:8000) and reload a few times until the GovReady-Q Compliance Server becomes available:



powered by GovReady Q

Open source compliance software for everyone to innovate securely

Stop suffering and start answering simple questions with your teammates. We'll help you identify security risks, learn compliance controls, and prepare required documents.

Setting up your organization and administrative user

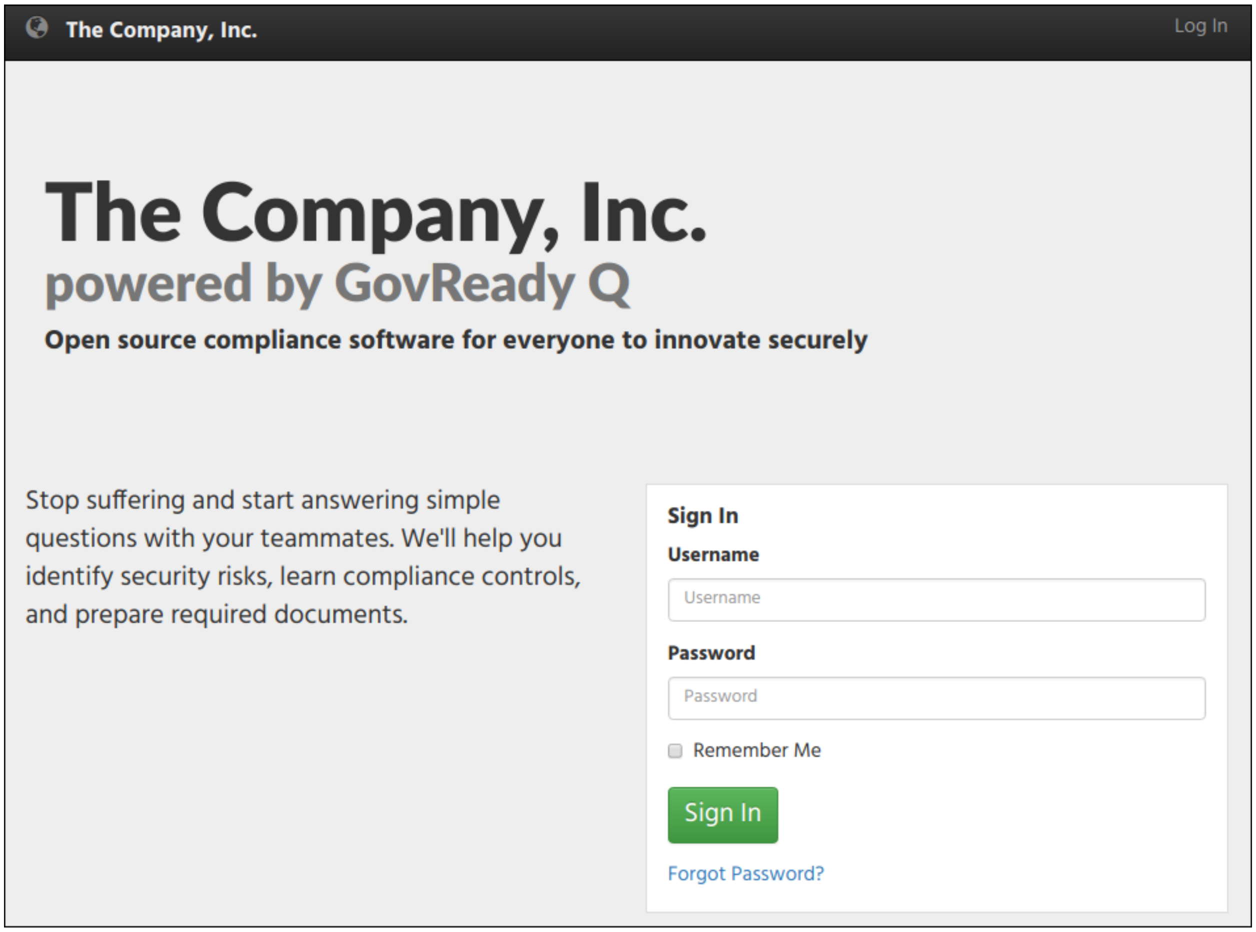
Now that the GovReady-Q Compliance Server is running, create an administrative account and an organization. Run the following command and answer the prompts:

```
docker container exec -it govready-q first_run
```

Your prompt and reply will look something like this:

```
Installed 2 object(s) from 1 fixture(s)
Let's create your first Q user. This user will have superuser privileges in the Q␣
↪administrative interface.
Username: admin
Email address: admin@mycompany.com
Password:
Password (again):
Superuser created successfully.
Let's create your Q organization.
Organization Name: The Company, Inc.
```

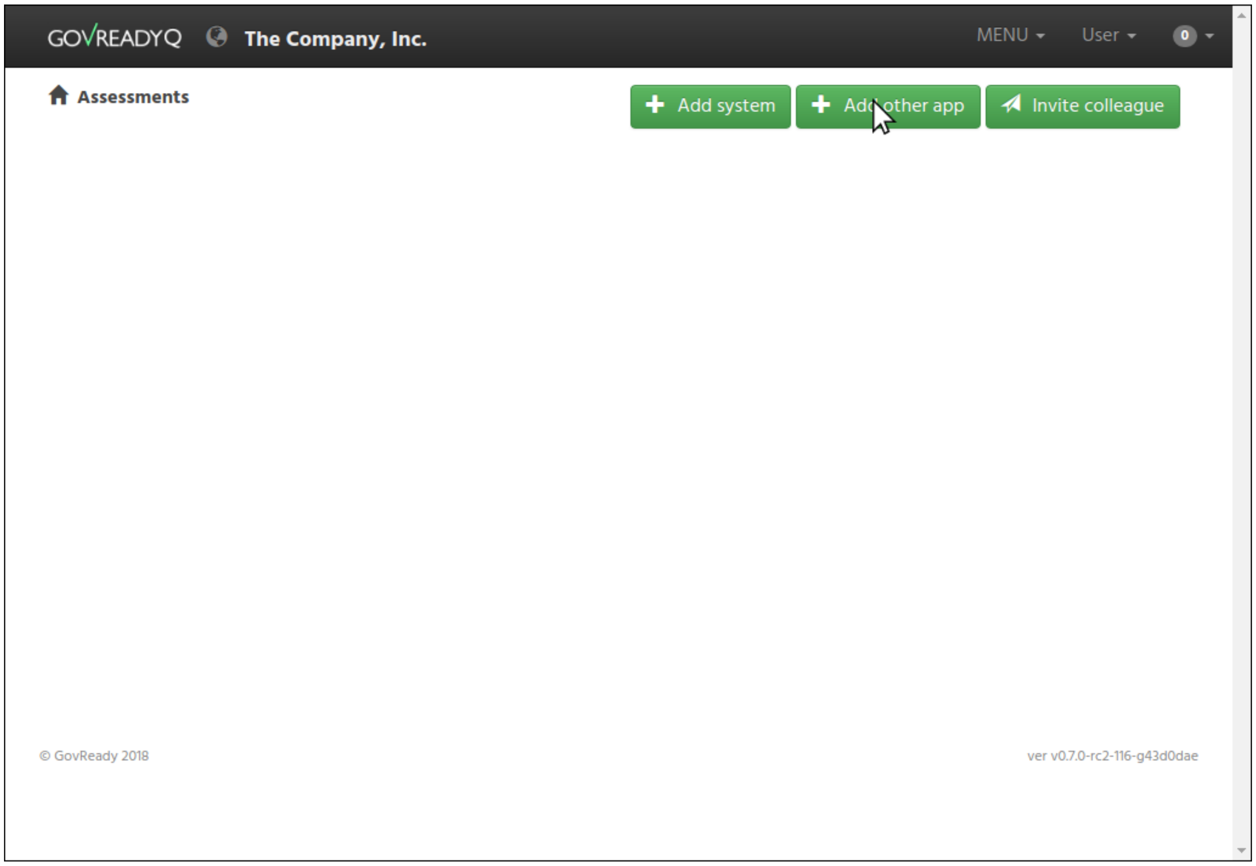
Now return to your browser, reload the page, and notice the company name has updated:



The Company, Inc.
powered by GovReady Q
Open source compliance software for everyone to innovate securely

Stop suffering and start answering simple questions with your teammates. We'll help you identify security risks, learn compliance controls, and prepare required documents.

You can now sign in with the administrative username and password you created.

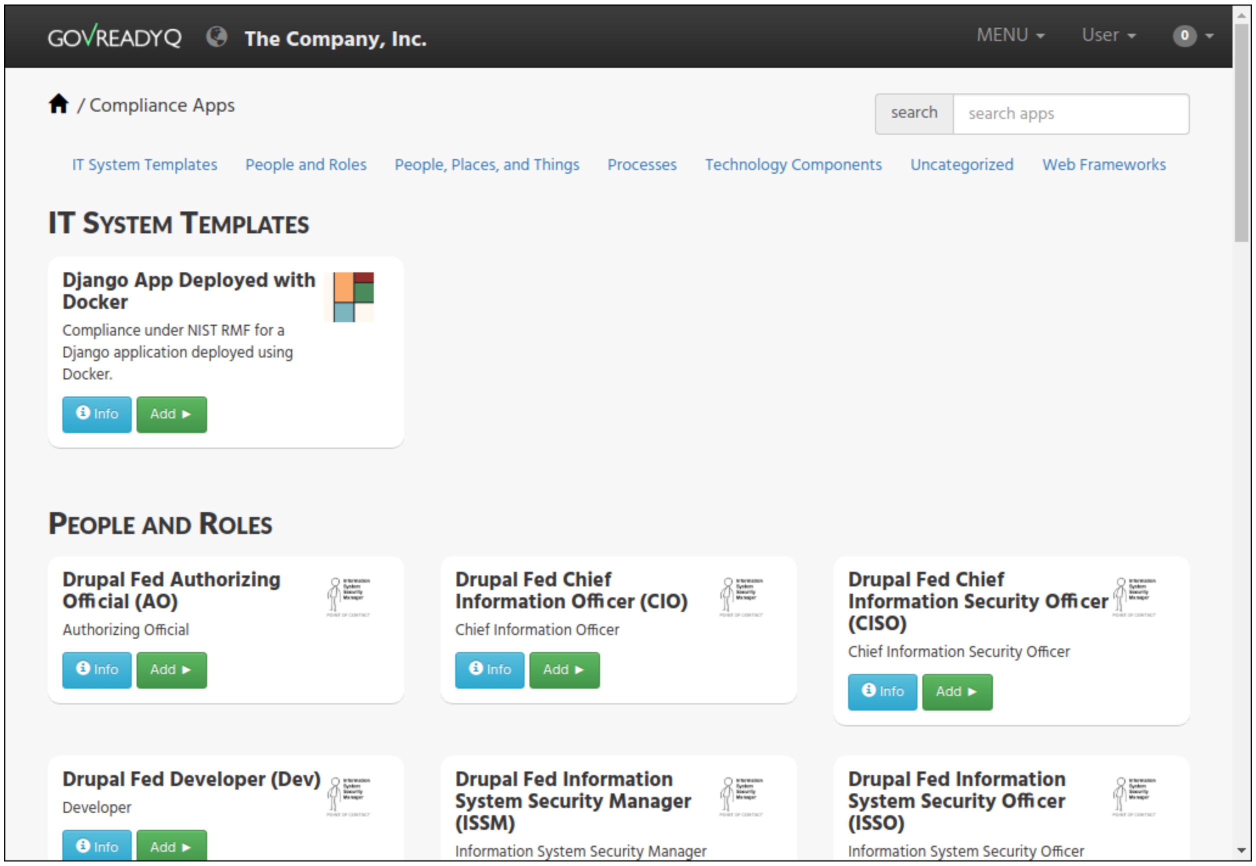


Congratulations! You've installed GovReady-Q Compliance Server configured for local development of compliance apps!

4.2.3 Step 3: Creating a compliance app

Creating the app

In this section we'll create our first compliance app. The app will appear in the compliance apps catalog in GovReady-Q. Click `Add other app` in your browser to go to the compliance apps catalog.



IT SYSTEM TEMPLATES

Django App Deployed with Docker
Compliance under NIST RMF for a Django application deployed using Docker.

PEOPLE AND ROLES

Drupal Fed Authorizing Official (AO)
Authorizing Official

Drupal Fed Chief Information Officer (CIO)
Chief Information Officer

Drupal Fed Chief Information Security Officer (CISO)
Chief Information Security Officer

Drupal Fed Developer (Dev)
Developer

Drupal Fed Information System Security Manager (ISSM)
Information System Security Manager

Drupal Fed Information System Security Officer (ISSO)
Information System Security Officer

Let's create our first compliance app! Use the command below:

```
docker container exec -it govready-q ./manage.py compliance_app host myfirstapp
```

The output will be:

```
Created new app in AppSource host at /usr/src/app/q-files/myfirstapp
```

The path shown in the output is a path *within* the container's filesystem, which is inaccessible from the workstation. The actual path is inside the path given to the `--appsdevdir` command line argument previously. If you followed our steps above exactly, you can see the app's files in your `apps` folder:

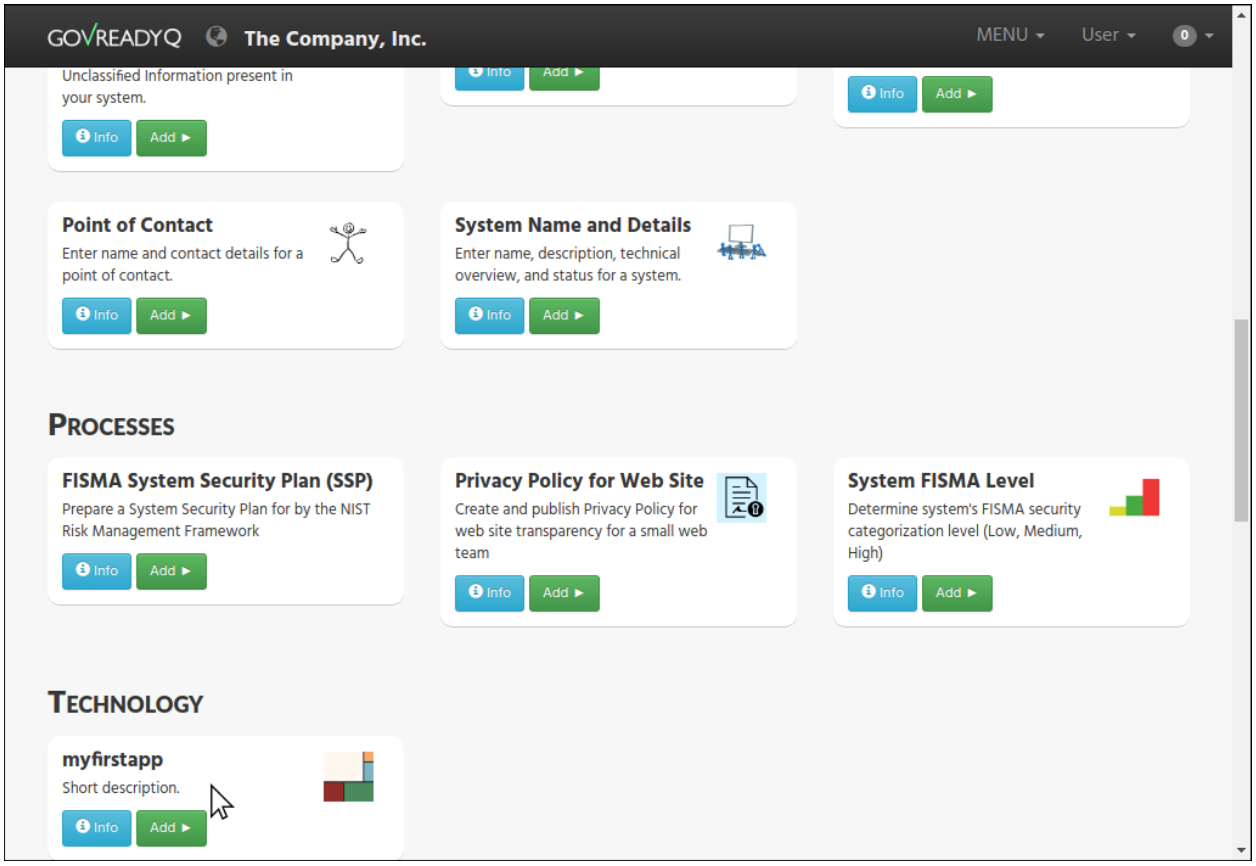
```
$ ls -l apps/myfirstapp
-rw-r--r-- 1 root root  664 Oct 25 11:43 app.yaml
drwxr-xr-x 2 root root 4096 Oct 25 11:43 assets
-rw-r--r-- 1 root root  449 Oct 25 11:43 example.yaml
```

Head back to your browser and reload the compliance apps catalog page.

Your new app may not appear because the catalog is cached. To clear the cache, restart the container:

```
docker container restart govready-q
```

After a few moments the container will be back up. Reload the compliance app catalog page. You should now see your app if you scroll to the end:


GOVREADYQ
The Company, Inc.
MENU
User
0
Unclassified Information present in your system.
Info
Add ▸
Point of Contact
Enter name and contact details for a point of contact.
System Name and Details
Enter name, description, technical overview, and status for a system.
PROCESSES
FISMA System Security Plan (SSP)
Prepare a System Security Plan for by the NIST Risk Management Framework
Privacy Policy for Web Site
Create and publish Privacy Policy for web site transparency for a small web team
System FISMA Level
Determine system's FISMA security categorization level (Low, Medium, High)
TECHNOLOGY
myfirstapp
Short description.

The development directory on the workstation now holds:

```
.
├── apps
│   └── myfirstapp
│       ├── app.yaml
│       ├── assets
│       │   └── app.png
│       └── example.yaml
├── database.sqlite
└── docker_container_run.sh
```

(More information about the structure of the app directory can be found in Understanding Compliance Apps.)

Editing app catalog metadata

Open `apps/myfirstapp/app.yaml` in a text editor. Edit the short description and add some text describing the app you are building:

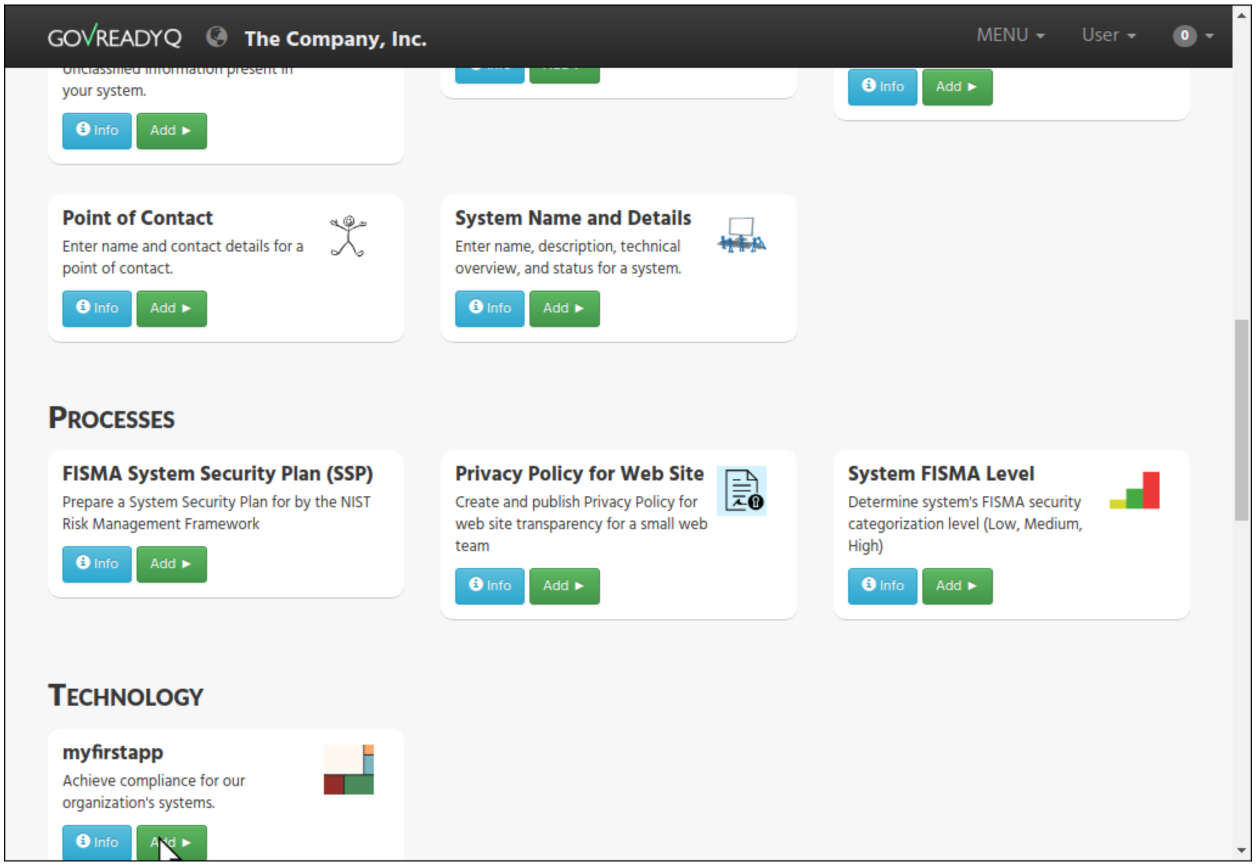
```
description:
  short: |
    Achieve compliance for our organization's systems.
```

Since this file was created by Docker, which is running as root, the file will be owned by root. You may need to use sudo to edit this file.

Reload the container to clear the app catalog cache:

```
docker container restart govready-q
```

And then reload the catalog page in your browser to see your description beneath `myfirstapp`. You can also edit the app's title and other catalog metadata, including the app's icon in `apps/myfirstapp/assets/app.png`.



your system.

Point of Contact
Enter name and contact details for a point of contact.

System Name and Details
Enter name, description, technical overview, and status for a system.

PROCESSES

FISMA System Security Plan (SSP)
Prepare a System Security Plan for by the NIST Risk Management Framework

Privacy Policy for Web Site
Create and publish Privacy Policy for web site transparency for a small web team

System FISMA Level
Determine system's FISMA security categorization level (Low, Medium, High)

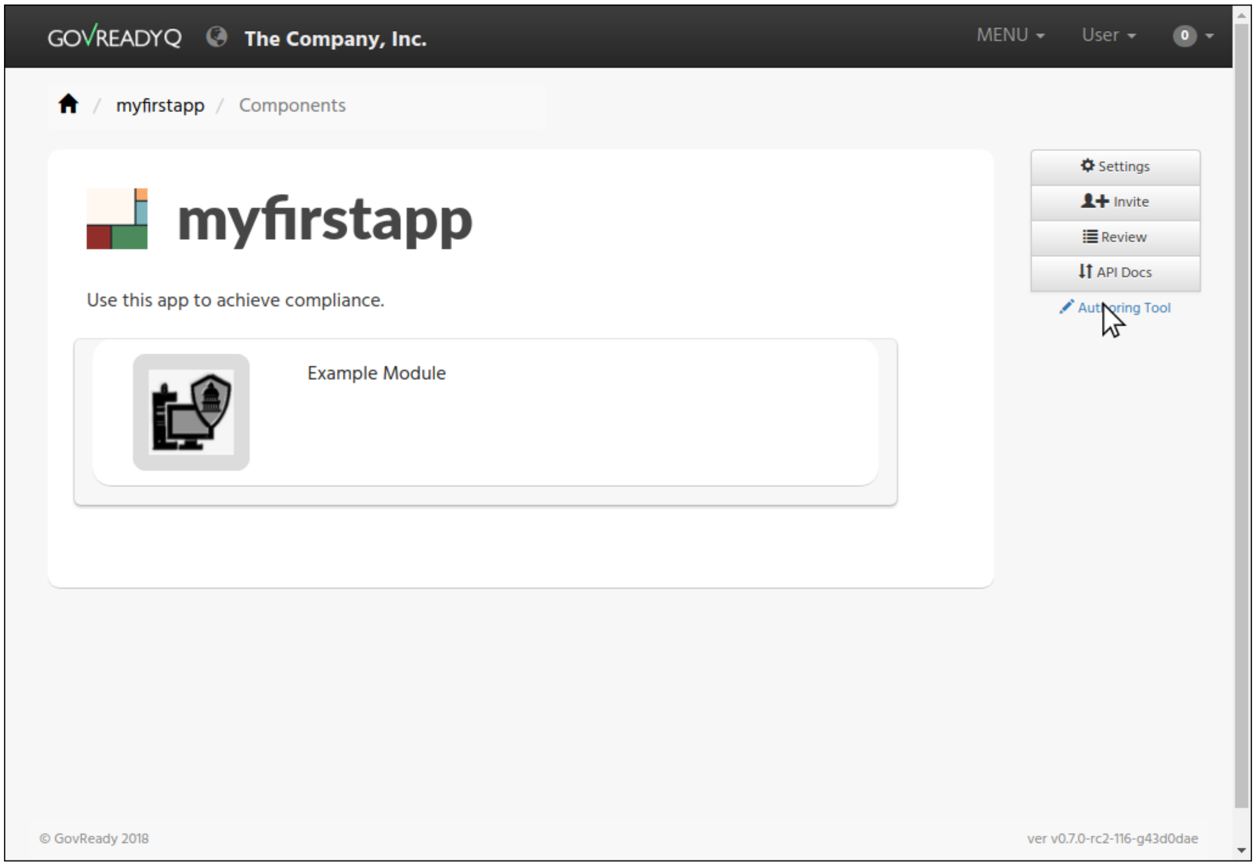
TECHNOLOGY

myfirstapp
Achieve compliance for our organization's systems.

4.2.4 Step 4: Edit the compliance app's YAML files

Start the app

In your browser, click on the `myfirstapp` entry's Add button in the app catalog.



About editing the app

We can edit our new compliance app by editing its `app.yaml` and `example.yaml` files on disk in our favorite text editor (described in this section) or with GovReady-Q's built-in authoring tools (described in the next section).

After each edit to the compliance apps files on disk, it may be necessary to restart the Docker container if you modified app catalog metadata (as you did above with `docker container restart`) or start a new instance of the compliance app from the compliance apps catalog page in your browser, if you modified the app's questions and output templates.

GovReady-Q purposely does not automatically recognize changes to compliance apps on disk until a new instance of the app is selected or a reload command (described below) is issued. This ensures previously loaded versions of the compliance app correctly maintain data entered by end-users.

Editing the app's main page

The opening screen of the app is determined by the `questions` section of the `app.yaml` file:

```
questions:
- id: q1
  title: Example Module
  type: module
  module-id: example
```

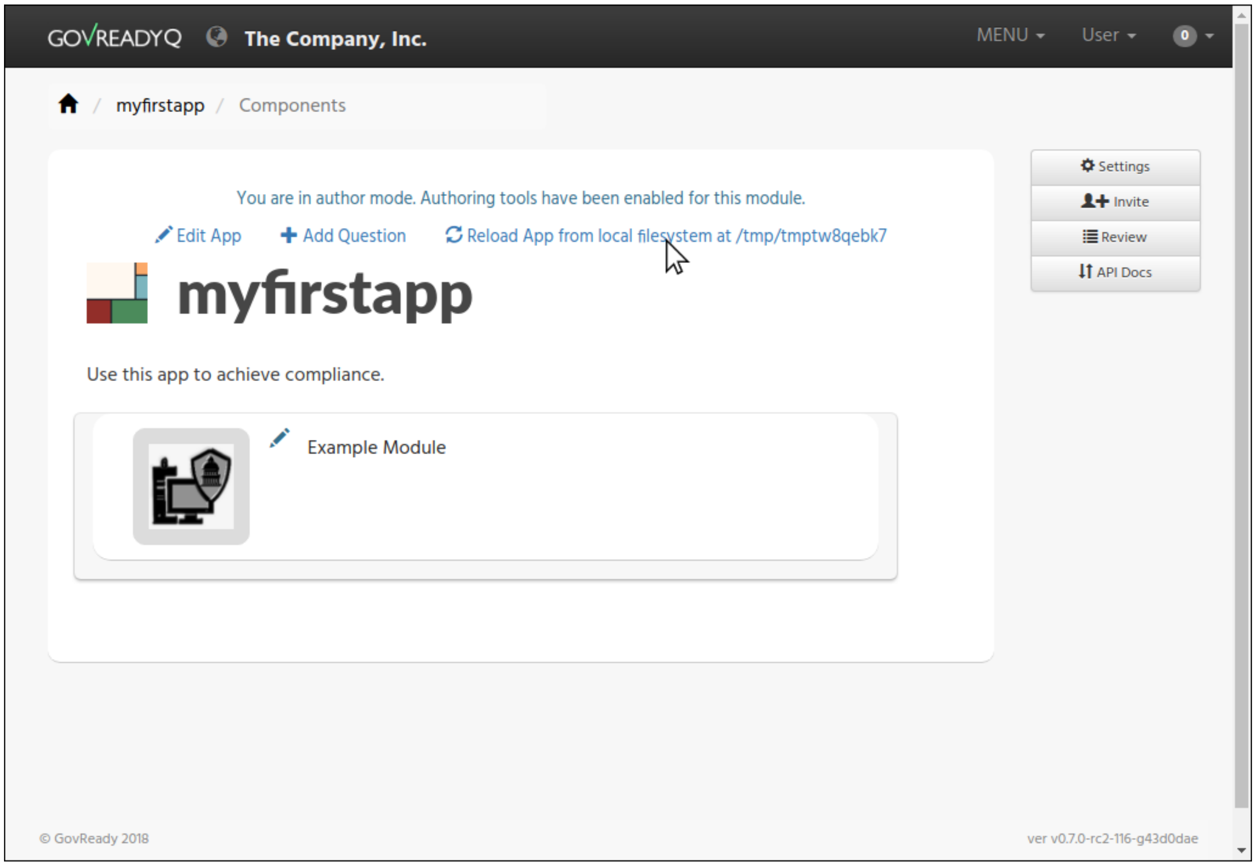
The new app has a single question labeled by the title `Example Module`, as you see in the YAML and in your browser. When the user clicks Example Module in the browser, they will start a new module defined by the YAML file referenced in the `module-id` data, in this case `example.yaml`.

Edit the `title` to:

```
title: Start Compliance
```

As described above, reloading the page in the browser will not show the change. This is by design. Since you are developing an app on your local filesystem, the GovReady-Q authoring tools are available.

Click `Authoring Tool` in the right column, and then click `Reload App from local filesystem`. (Alternatively, you could return to the compliance app catalog page and add the app again.)



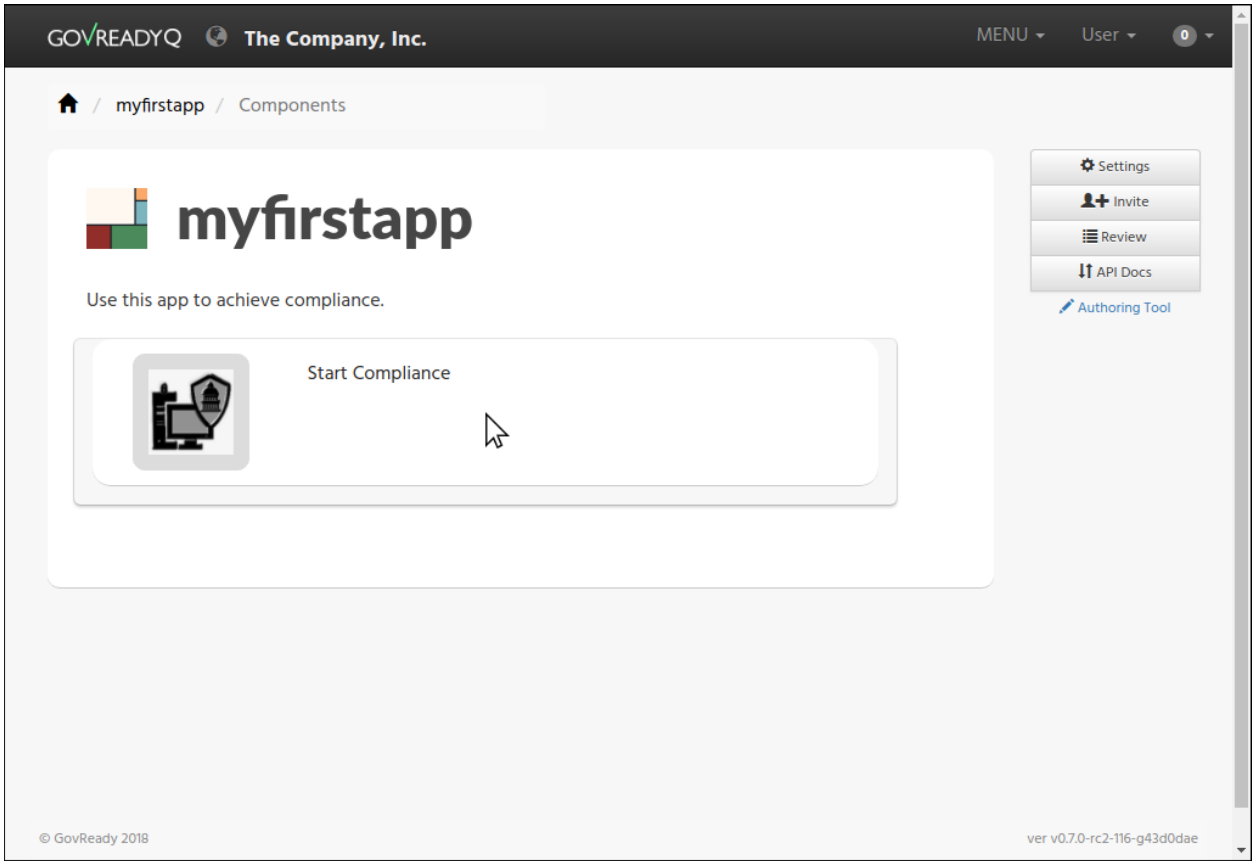
You are in author mode. Authoring tools have been enabled for this module.

Reload App from local filesystem at /tmp/tmptw8qebk7

myfirstapp

Use this app to achieve compliance.

Note how `Start Compliance` now appears in the browser.

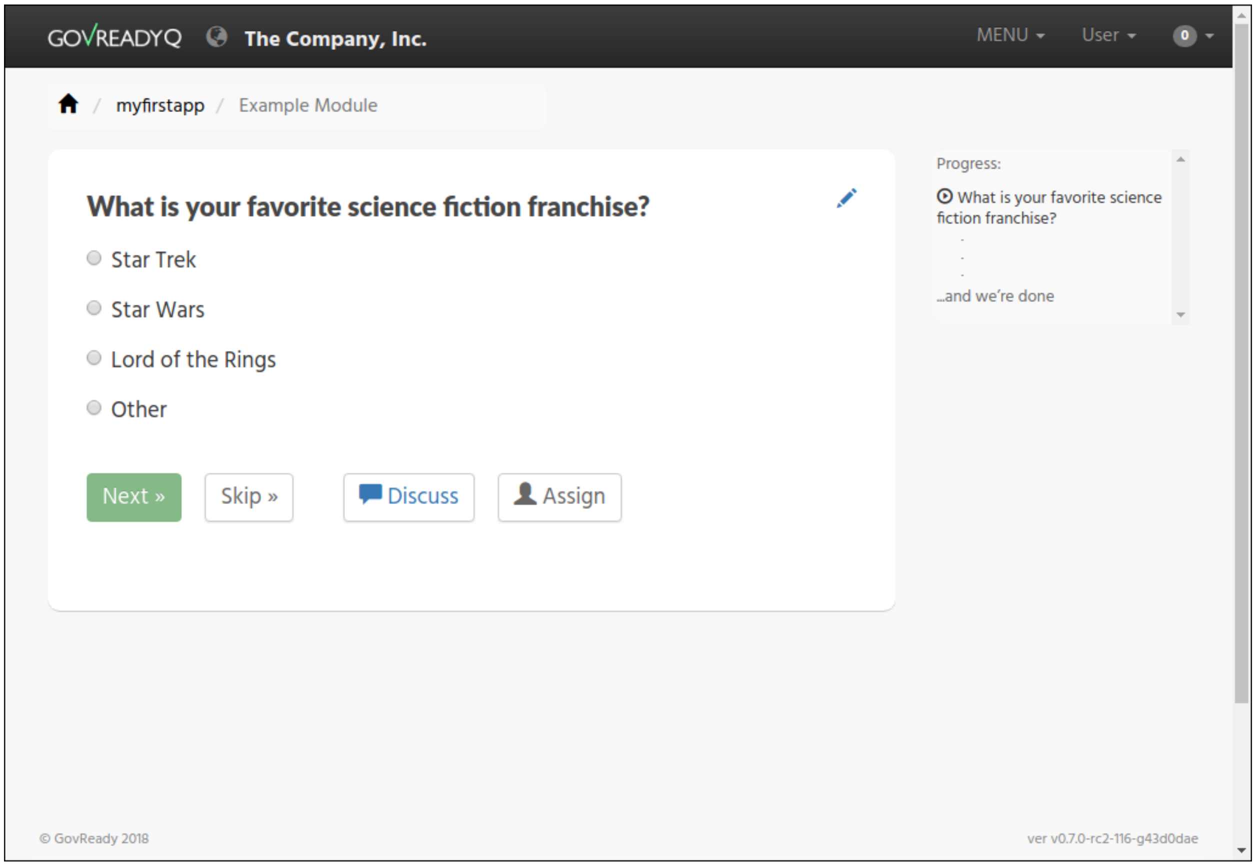


myfirstapp

Use this app to achieve compliance.

Editing the app's first module

Click `Start Compliance`. This begins the app's module defined in `example.yaml`. The example module contains a single sample question:


GOVREADYQ
The Company, Inc.
MENU
User
0
myfirstapp / Example Module
Progress:
What is your favorite science fiction franchise?
...and we're done
Next »
Skip »
Discuss
Assign
© GovReady 2018
ver v0.7.0-rc2-116-g43d0dae

What is your favorite science fiction franchise?

- Star Trek
- Star Wars
- Lord of the Rings
- Other

Open `example.yaml` and see that the question's type, prompt, and choices are defined in the YAML file's `question`'s section:

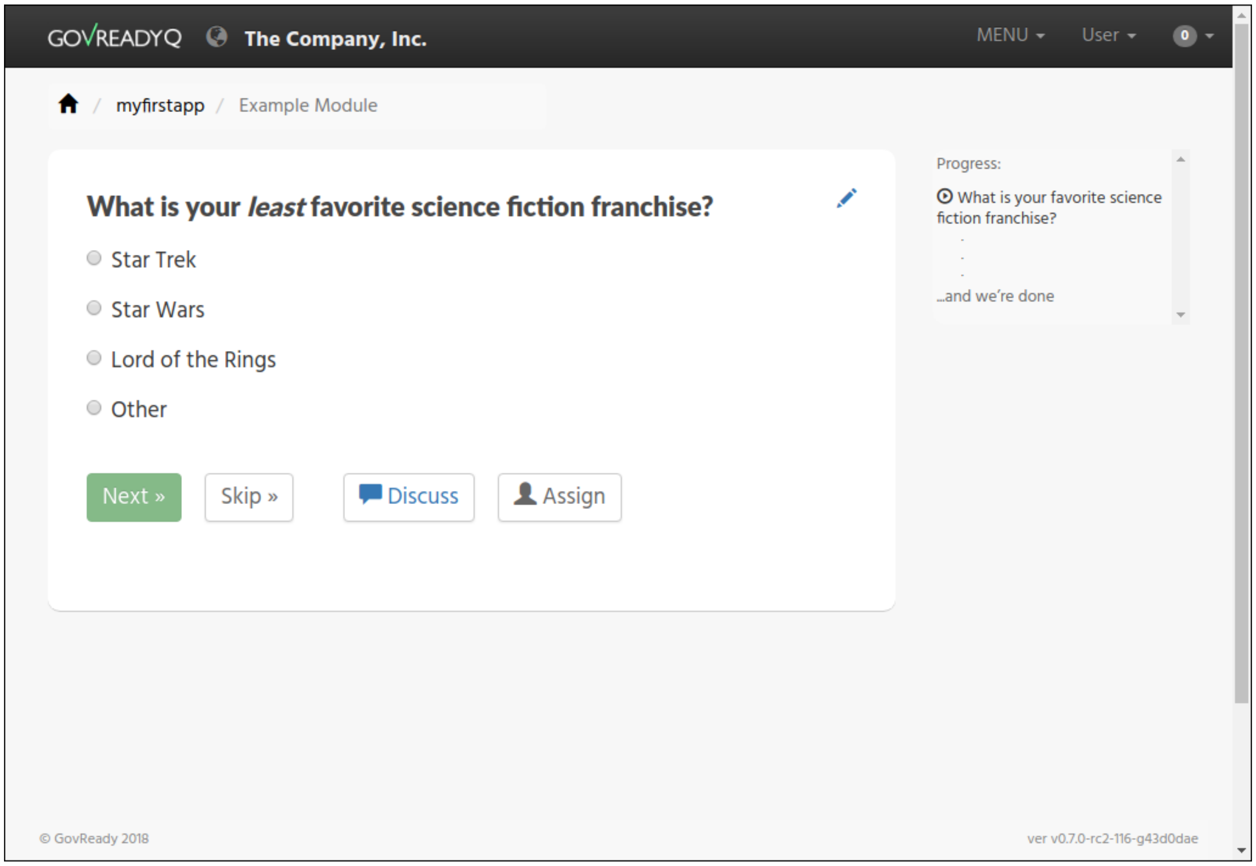
```
questions:
- id: q1
  title: What is your favorite science fiction franchise?
  prompt: What is your favorite science fiction franchise?
  type: choice
  choices:
    - key: startrek
      text: Star Trek
    - key: starwars
      text: Star Wars
    - key: lordoftherings
      text: Lord of the Rings
    - key: other
      text: Other
```

Change the `prompt` or `choices`.

(As with `app.yaml`, since this file was created by Docker the file will be owned by root. You may need to use sudo to edit this file.)

As described above, reloading the page in the browser will not show the change. This is by design. Go back to the main app page, click `Authoring Tool` and then `Reload App from local filesystem`, and then go back to the Start Compliance page.

Your changes are now seen in your browser.



What is your *least* favorite science fiction franchise?

- Star Trek
- Star Wars
- Lord of the Rings
- Other

Progress:
What is your favorite science fiction franchise?
.
.
.
...and we're done

More information about the file format of modules can be found in Modules, Questions, and Documents.

4.2.5 Step 5: Edit a compliance app using GovReady-Q's authoring tools

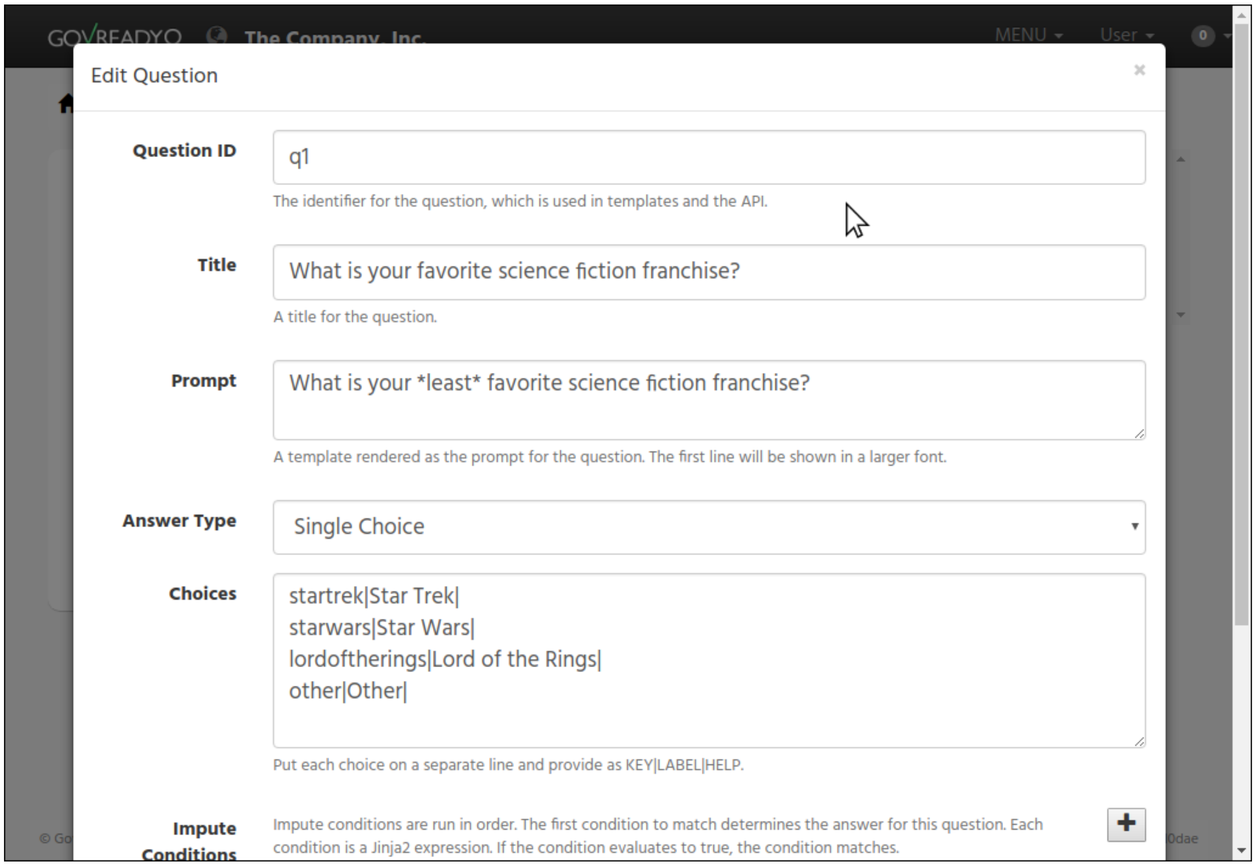
About the authoring tools

It is also possible to edit a compliance app's questions without leaving your browser. When editing the compliance app via GovReady-Q's built-in authoring tools, you will immediately see the changes in the instance of the compliance app you are editing without having to reload it. The changes are also immediately written to the files on disk.

GovReady-Q's built-in authoring tools will let you edit and add questions, but currently won't let you change the name of the description of the app in catalog. You will still need to edit those details in the compliance app YAML files stored on disk, as described above.

Editing a question

A blue pencil icon will appear at the top right of module questions when the authoring tools are available. Click the pencil icon for the sample question. The question editor will pop up:



Edit Question

Question ID: q1
The identifier for the question, which is used in templates and the API.

Title: What is your favorite science fiction franchise?
A title for the question.

Prompt: What is your *least* favorite science fiction franchise?
A template rendered as the prompt for the question. The first line will be shown in a larger font.

Answer Type: Single Choice

Choices:
```
startrek|Star Trek|
starwars|Star Wars|
lordoftherings|Lord of the Rings|
other|Other|
```
Put each choice on a separate line and provide as KEY|LABEL|HELP.

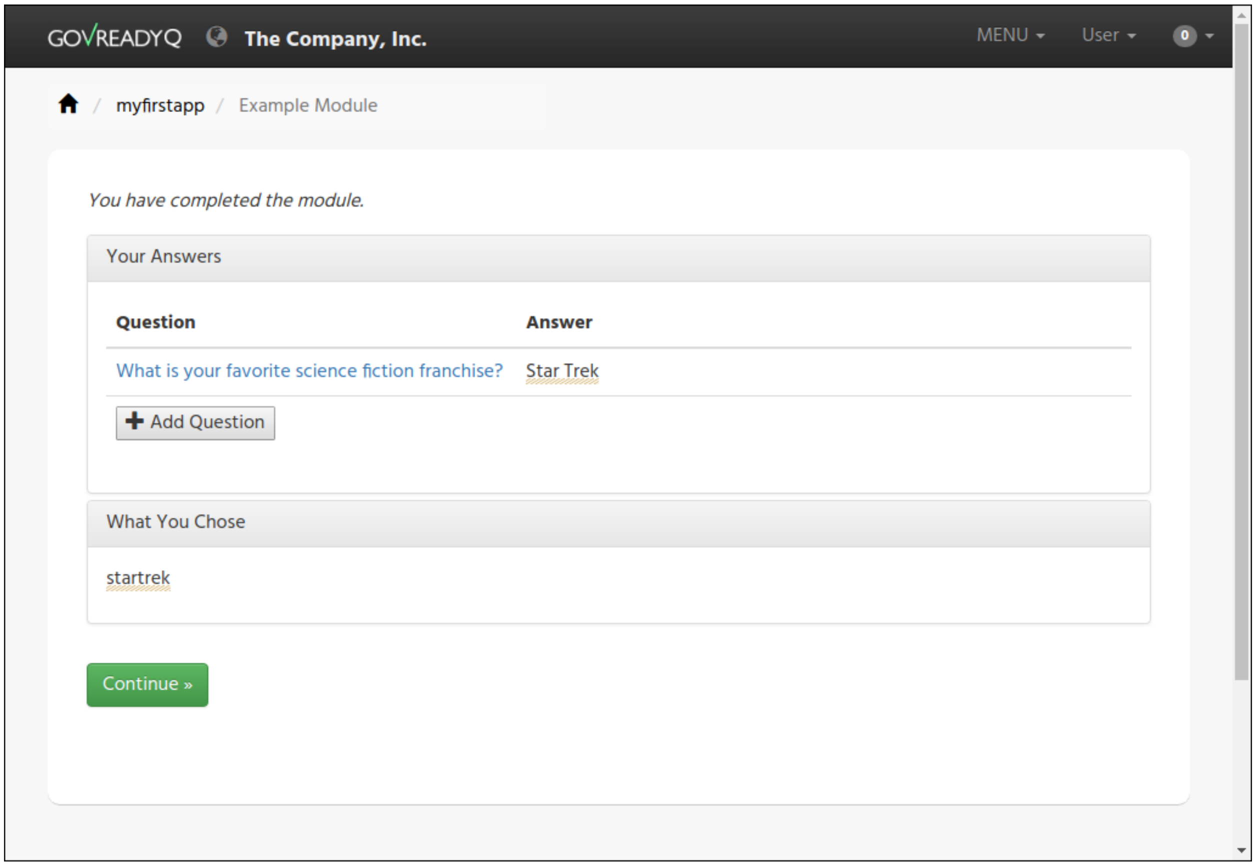
Impute Conditions: Impute conditions are run in order. The first condition to match determines the answer for this question. Each condition is a Jinja2 expression. If the condition evaluates to true, the condition matches.

This is a much easier way of editing questions! Try editing this question. After clicking `Save Changes`, look in your text editor to see that the changes have been immediately saved to `example.yaml`.

Adding questions

It is also possible to add questions. In order to add a question, all of the existing questions must be answered. Answer the sample question, or click `Skip`.

You'll see an `Add Question` button on the module review page:



You have completed the module.

Your Answers

Question	Answer
What is your favorite science fiction franchise?	Star Trek

What You Chose

startrek

Try out the Add Question button now. It will create a new text question. Use the blue pencil icon to change the question's prompt and choices.

You have now seen how to create and edit an app!

4.2.6 Step 6: Deploy the app to a production instance of GovReady-Q

Adding apps to a git repository

Your workstation's instance of GovReady-Q has been configured to load apps from the local filesystem. Your organization's production instance of GovReady-Q can be configured similarly, but more likely it will be configured to load apps from a remote git repository.

Create a new git repository in your source code control system and push your `apps` directory to the repository. The repository's root directory should contain a directory named `myfirstapp`:

```
repository root
  └── myfirstapp
        ├── app.yaml
        ├── assets
        │   └── app.png
        └── example.yaml
```

If you have an existing source code control system containing apps in this layout, consider checking out the repository locally so that it is in the same path provided to the `--appsdevdir` argument to `docker_container_run.sh`. If your repository is in a different layout or if you are using multiple repositories to store compliance apps, see below.

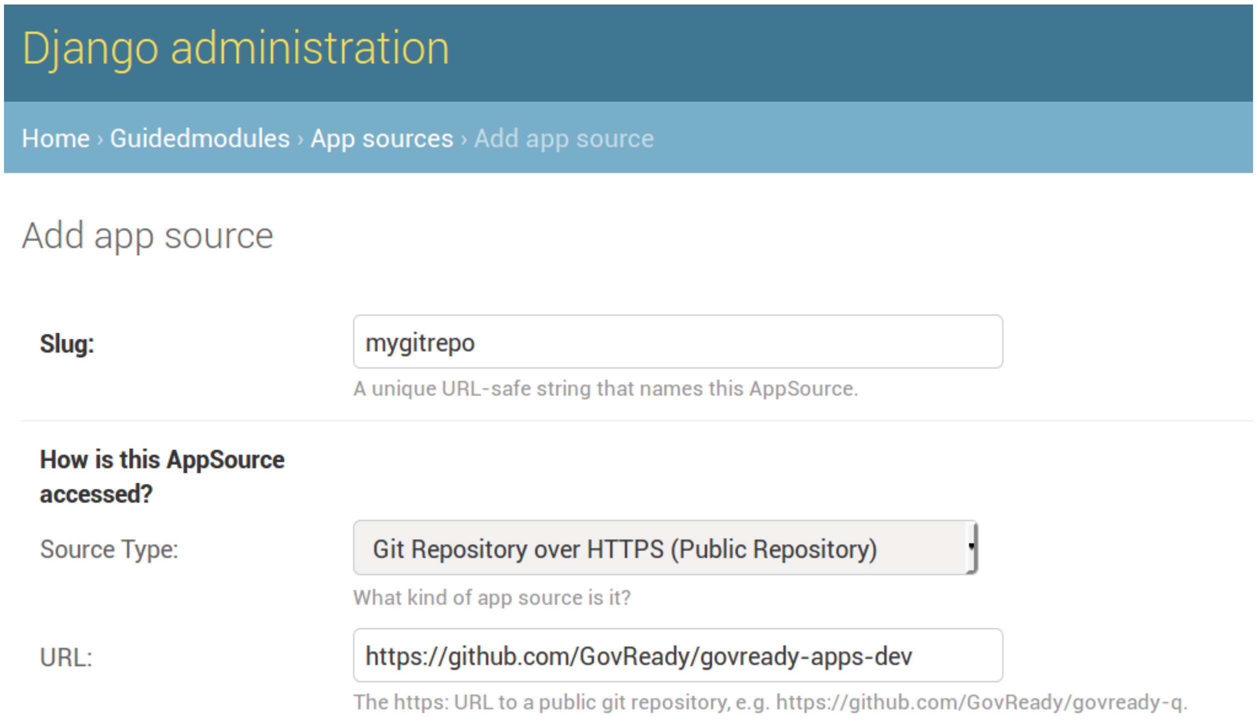
Configuring a production system to load apps from the git repository

On the production GovReady-Q instance, log into the Django admin at `https://production-q/admin`. Add a new App Source.

Set its `Slug` to a short name for the repository, composed of letters, numbers, and underscores, such as `mygitrepo`.

If your git repository is public or accessible over an https: URL

If your git repository is accessible over an https: URL (such as a public GitHub repository), change the Source Type to Git Repository over HTTPS and paste the URL into the URL field. The other fields can be left blank. Here's what that looks like:



Add app source

Slug: mygitrepo
A unique URL-safe string that names this AppSource.

How is this AppSource accessed?

Source Type: Git Repository over HTTPS (Public Repository)
What kind of app source is it?

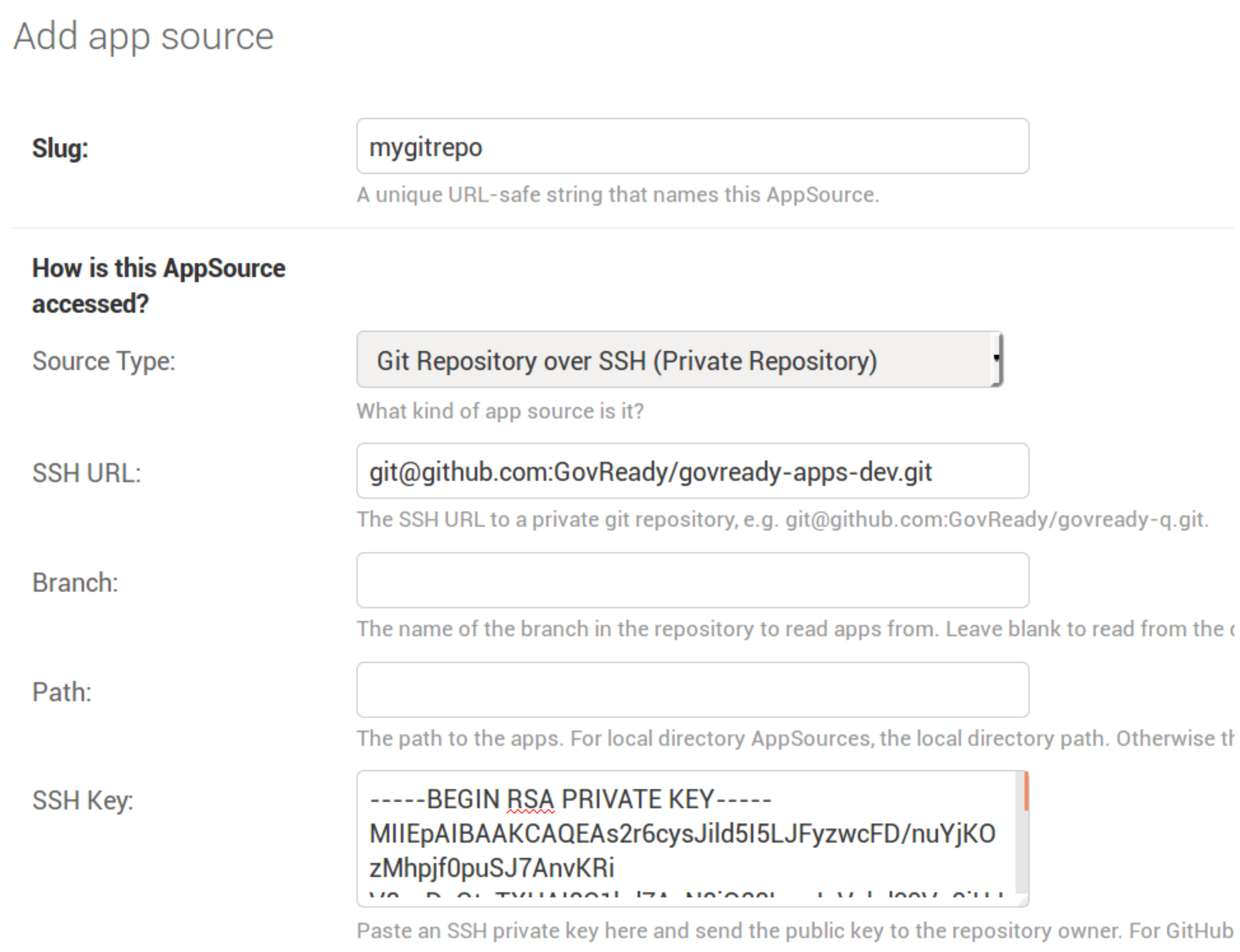
URL: https://github.com/GovReady/govready-apps-dev
The https: URL to a public git repository, e.g. https://github.com/GovReady/govready-q.

If your git repository is private

If your git repository is private and accessible instead using an SSH URL (typically git@github.com:organization/repository.git) and an SSH public/private keypair, such as with GitHub or GitLab deploy keys, then first create a new SSH key for your GovReady-Q instance:

```
ssh-keygen -q -t rsa -b 2048 -N "" -C "_your-repo-name_-deployment-key" -f ./repo_
↪deploy_key
```

Your GovReady-Q instance will hold the private key half of the newly generated keypair, and your source code control system will hold the public key. Back in the Django admin, set the Source Type to Git Repository over SSH. Paste the SSH URL into the URL field. Then open the newly generated file `repo_deploy_key` and paste its contents into the SSH Key field. The other fields can be left blank. Here's what that looks like:


Add app source

Slug: mygitrepo

A unique URL-safe string that names this AppSource.

How is this AppSource accessed?

Source Type: Git Repository over SSH (Private Repository)

What kind of app source is it?

SSH URL: git@github.com:GovReady/govready-apps-dev.git

The SSH URL to a private git repository, e.g. git@github.com:GovReady/govready-q.git.

Branch:

The name of the branch in the repository to read apps from. Leave blank to read from the

Path:

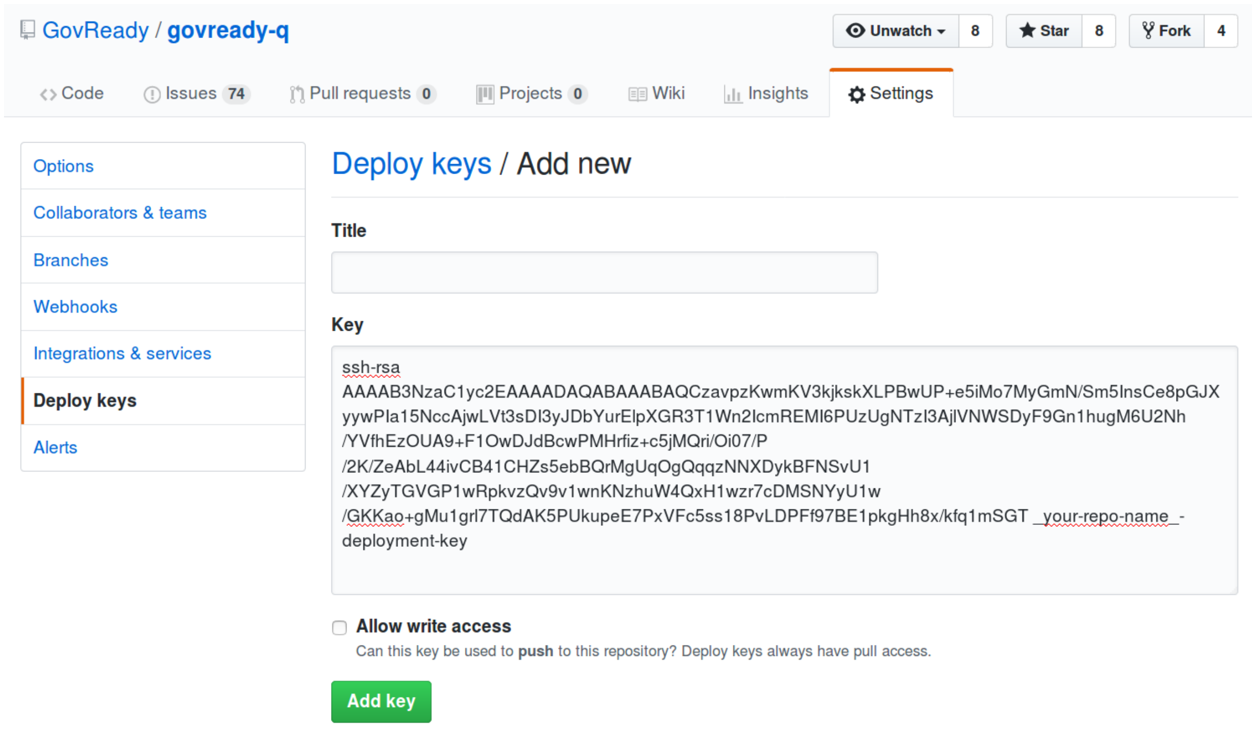
The path to the apps. For local directory AppSources, the local directory path. Otherwise th

SSH Key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAs2r6cysJild5I5LJFyzwcFD/nuYjKO
zMhpjf0puSJ7AnvKRi
```

Paste an SSH private key here and send the public key to the repository owner. For GitHub

Copy the public key in the newly generated file `repo_deploy_key.pub` into the deploy keys section of your source code repository. Here is what that looks like on GitHub:



Deploy keys / Add new

Title

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCzavpzKwmKV3kjkskXLPBwUP+e5iMo7MyGmN/Sm5InsCe8pGJX
yywPIa15NccAjwLVt3sDI3yJDbYurElpXGR3T1Wn2IcmREMI6PUzUgNTzI3AjlVNWSDyF9Gn1hugM6U2Nh
/YVfhEzOUA9+F1OwDJdBcwPMHrfiz+c5jMQri/Oi07/P
/2K/ZeAbL44ivCB41CHZs5ebBQrMgUqOgQqqzNNXDykBFNSvU1
/XYZyTGVGP1wRpkvzQv9v1wnKNzhuW4QxH1wzr7cDMSNYyU1w
/GKKao+gMu1grl7TQdAK5PUkupeE7PxVFc5ss18PvLDPFf97BE1pkgHh8x/kfq1mSGT _your-repo-name_-
deployment-key
```

☐ **Allow write access**

Can this key be used to **push** to this repository? Deploy keys always have pull access.

Other information about App Sources

As with local development, the production system's compliance app catalog may be cached. To see new apps, restart the production instance of GovReady-Q.

See App Sources for more information about how to configure your production instance of GovReady-Q to load apps from local filesystem directories, git repositories (including on-prem git repositories), or GitHub.

Advanced setups for development with a repository of apps

In this guide we have used the `--appsdevdir` command to specify a location in which app YAML files and assets are stored. In a small setup, all apps could be stored in a subdirectory of the location given to `--appsdevdir`. But you may want to separate apps into different folders, such as if they are divided between folders in a single git repository or across multiple git repositories, then a more advanced configuration of GovReady-Q is necessary.

Imagine the following directory structure where two GitHub repositories are cloned into two separate local directories within `apps`, and each has a `compliance_apps` directory holding its apps:

```
.
├── apps (`--appsdevdir` directory)
│   ├── repo1
│   │   └── compliance_apps
│   │       ├── myfirstapp
│   │       ├── mysecondapp
│   └── repo2
│       └── compliance_apps
│           ├── mythirdapp
│           ├── myfourthapp
├── database.sqlite
└── docker_container_run.sh
```

The default setup from GovReady-Q docker installation only show apps in the compliance app catalog if the app files are located in the immediate subdirectory of path configured to load apps. But we can also tell GovReady-Q to load apps from *multiple* locations. In this case we will configure GovReady-Q to load apps from two locations:

```
apps/repo1/compliance_apps
apps/repo2/compliance_apps
```

Recall that the path given to `--appsdevdir` is mapped to a path within the Docker container so that the container can see the YAML files on the (host) local filesystem. The container sees these directories as

```
/usr/src/app/q-files/repo1/compliance_apps
/usr/src/app/q-files/repo2/compliance_apps
```

Log into the Django admin at `http://localhost:8000/admin`. Add two new `AppSource` entries:

For the first, set the `Slug` to `repo1` (or any other label that will help you distinguish the two repositories), the `Source Type` to `Local Directory`, and the `Path` to `/usr/src/app/q-files/repo1/compliance_apps`. For the second, set the `Slug` to `repo2`, the `Source Type` to `Local Directory`, and the `Path` to `/usr/src/app/q-files/repo2/compliance_apps`.

Then restart the container:

```
docker container restart govready-q
```

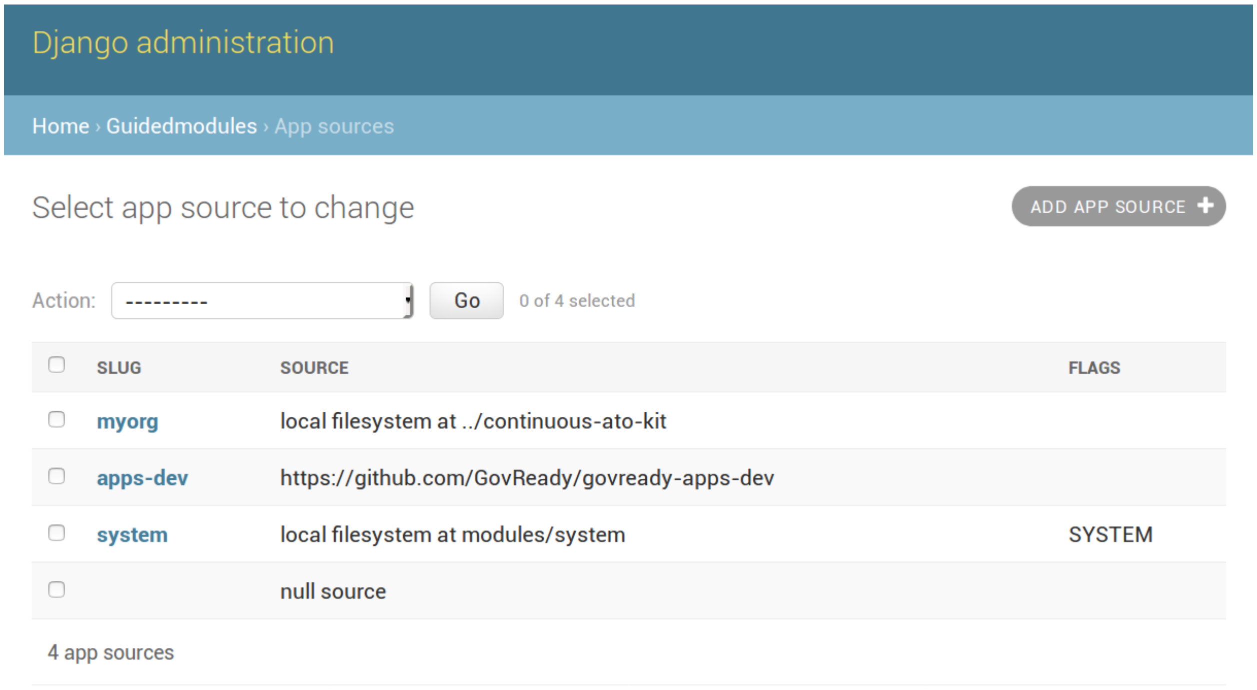
and the apps defined in all of the repositories should be visible in the compliance app catalog.

4.3 App Sources

GovReady-Q can be configured by an administrator to load compliance apps from one or more sources, which can be local directories or remote git repositories.

When using the Hosted Version of GovReady-Q, GovReady PBC is the administrator. If you are the administrator of an installation of GovReady-Q at your organization, the information below will help you configure the App Sources available to your users.

App Sources are configured in the Django admin at the URL `/admin` on your GovReady-Q domain under `App Sources`:



Django administration

Home › Guidedmodules › App sources

Select app source to change

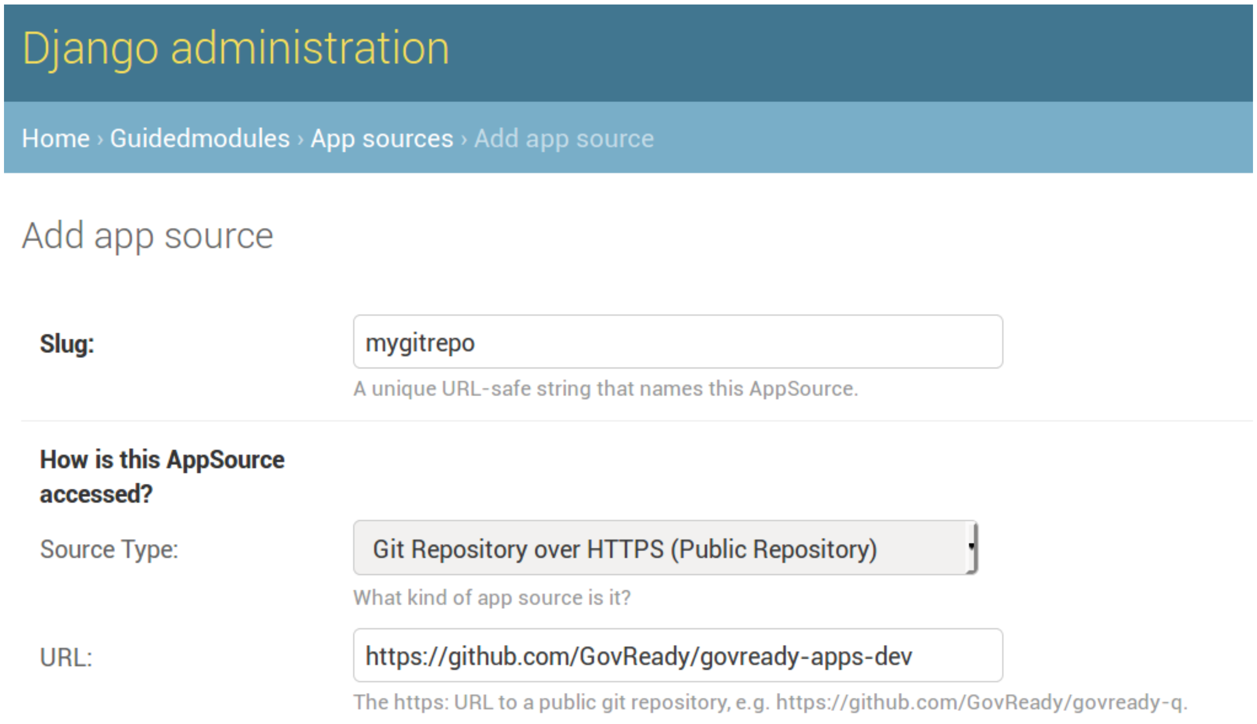
ADD APP SOURCE +

Action: --------- Go 0 of 4 selected

	SLUG	SOURCE	FLAGS
☐	myorg	local filesystem at ../continuous-ato-kit	
☐	apps-dev	https://github.com/GovReady/govready-apps-dev	
☐	system	local filesystem at modules/system	SYSTEM
☐		null source	

4 app sources

Each App Source points GovReady-Q to a directory or repository of compliance apps.



Django administration

Home › Guidedmodules › App sources › Add app source

Add app source

Slug: mygitrepo

A unique URL-safe string that names this AppSource.

How is this AppSource accessed?

Source Type: Git Repository over HTTPS (Public Repository)

What kind of app source is it?

URL: https://github.com/GovReady/govready-apps-dev

The https: URL to a public git repository, e.g. https://github.com/GovReady/govready-q.

4.3.1 App Source Slug

The first App Source field is the Slug. The Slug is a short name you assign to the App Source to distinguish it from other App Sources. The Slug is used to form URLs in GovReady-Q's compliance apps catalog, so it may only contain letters, numbers, dashes, underscores, and other URL path-safe characters.

4.3.2 App Source Type

There are four types of App Sources: local directories, remote git repositories using HTTP which are typically public repositories, remote git repositories using SSH which typically use SSH deploy keys for access, and remote GitHub repositories using a GitHub username and password for access.

Local Directory

The Local Directory source type directs GovReady-Q to load compliance apps from a directory on the same machine GovReady-Q is running on. (When deploying with Docker, that's on the container filesystem unless a path has been mounted to a volume or to the host machine.)

In the `Path` field, enter the path to a local directory containing compliance apps. This path is expected to contain a sub-directory for each compliance app contained in this source. For instance, if you have this directory layout:

```
.
└── home
    └── user
        └── compliance_apps
            ├── myfirstapp
            │   └── app.yaml
            └── mysecondapp
                └── app.yaml
```

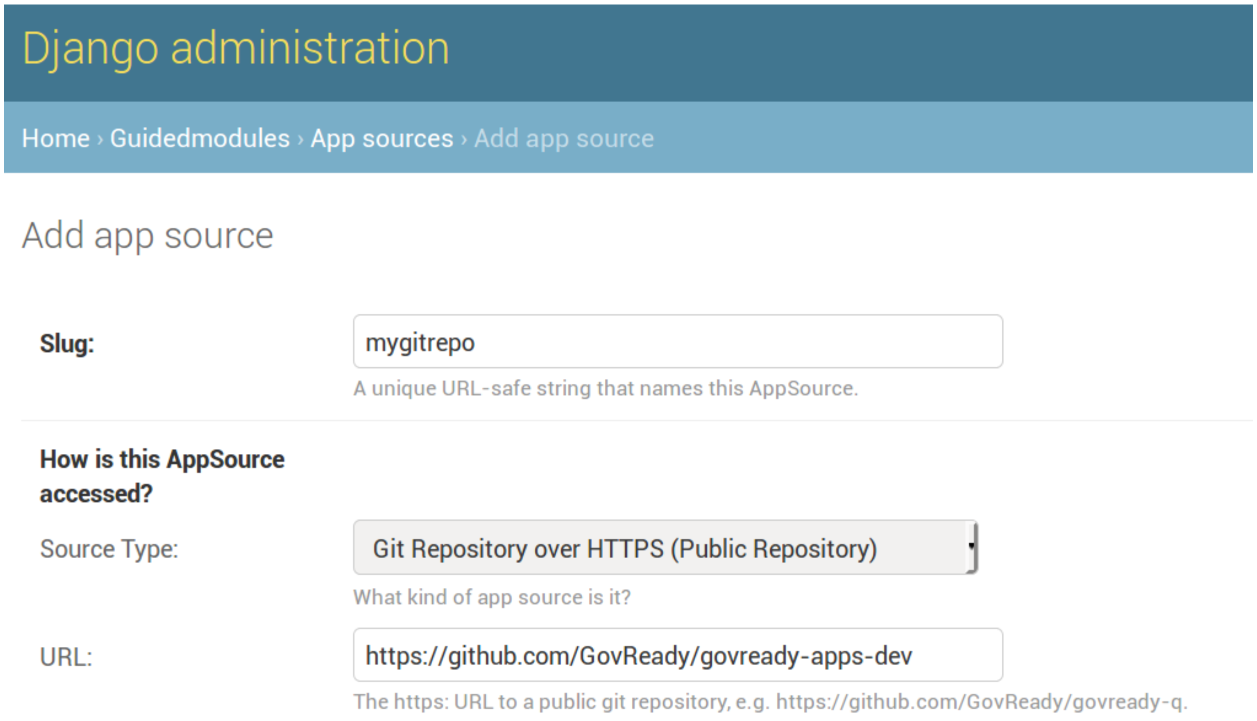
then your Path would be `/home/user/compliance_apps`.

The path can be absolute or relative to the path in which GovReady-Q is installed.

Git Repository over HTTPS

The Git Repository over HTTPS source type is for git repositories, such as on GitHub or GitLab, that can be cloned using an HTTPS URL. These repositories are typically public, or in an enterprise environment public within your organization's network.

Paste the HTTPS git clone URL — such as https://github.com/GovReady/govready-apps-dev — into the URL field. Here's what that looks like:



Add app source

Slug: mygitrepo

A unique URL-safe string that names this AppSource.

How is this AppSource accessed?

Source Type: Git Repository over HTTPS (Public Repository)

What kind of app source is it?

URL: https://github.com/GovReady/govready-apps-dev

The https: URL to a public git repository, e.g. https://github.com/GovReady/govready-q.

The other fields can be left blank.

The `Path` field optionally specifies a sub-directory within the repository in which the compliance apps are stored if they are not stored in the root of the repository. For instance if the repository has a directory layout similar to:

```
.
└── github.com/organization/repository
    └── apps
        ├── myfirstapp
        │   └── app.yaml
        └── mysecondapp
            └── app.yaml
```

then set the `Path` field to `apps`.

If the compliance apps are not in the repository's default branch (i.e. something other than the typical `master` default branch), then set the `Branch` field to the name of the branch to read the compliance apps from.

You can use HTTPS to access private repositories by placing your username and password or personal access token into the URL, such as:

```
https://username:password@github.com/GovReady/govready-apps-dev
```

Since this requires user credentials, it should be avoided for production deployments in favor of using Git Repository over SSH (see below).

Git Repository over SSH

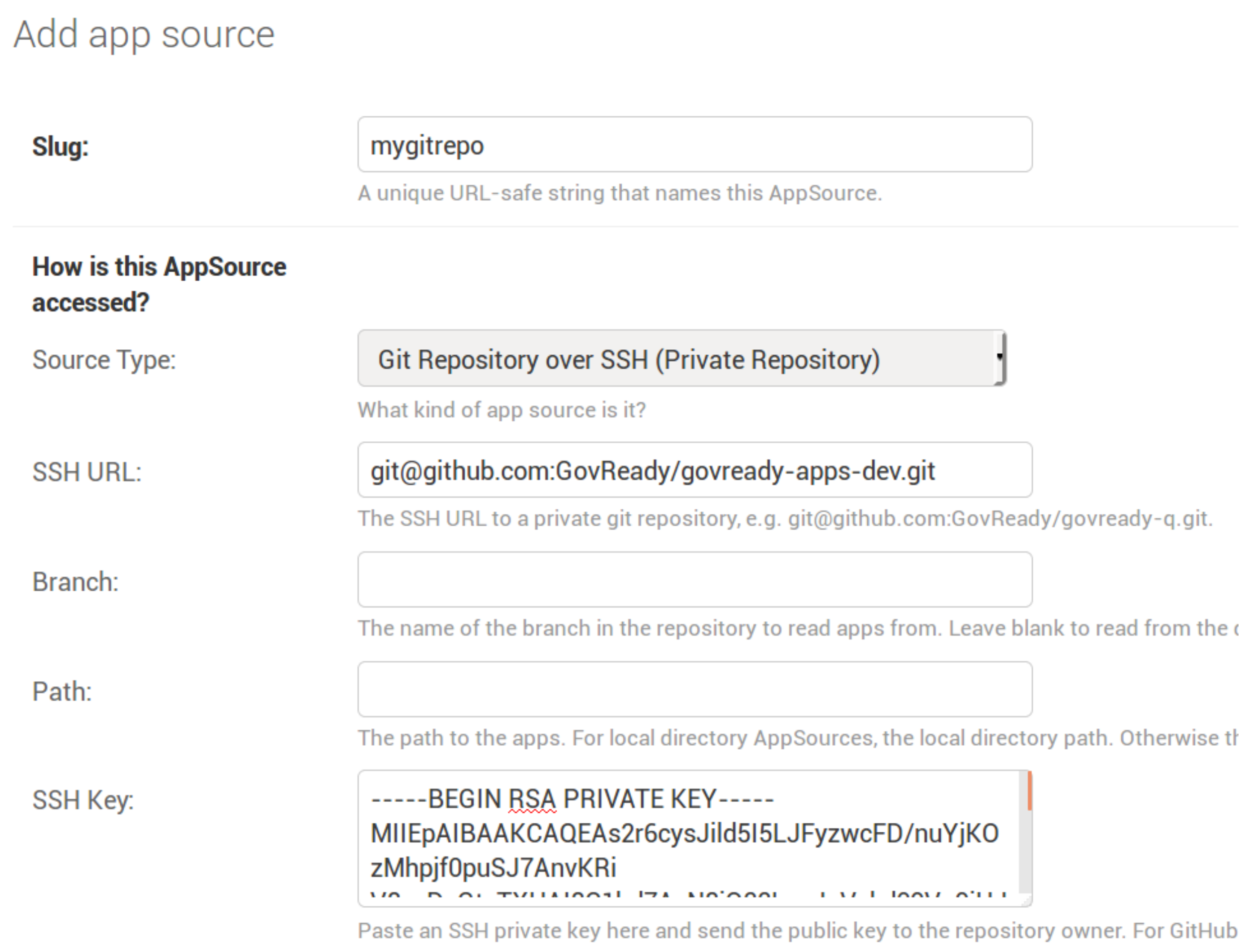
If your git repository is private and accessible using an SSH URL (which typically looks like git@github.com:organization/repository.git) and an SSH public/private keypair, such as with GitHub or GitLab deploy keys, then use the Git Repository over SSH source type.

Create a new SSH key for your GovReady-Q instance to be used as a deploy key:

```
ssh-keygen -q -t rsa -b 2048 -N "" -C "_your-repo-name_-deployment-key" -f ./repo_
↪deploy_key
```

Your GovReady-Q instance will hold the private key half of the newly generated keypair, and your source code control system will hold the public key.

Back in the Django admin, set the Source Type to Git Repository over SSH. Paste the git clone SSH URL into the URL field. Then open the newly generated file `repo_deploy_key` and paste its contents into the SSH Key field. Here's what that looks like:



Add app source

Slug: mygitrepo

A unique URL-safe string that names this AppSource.

How is this AppSource accessed?

Source Type: Git Repository over SSH (Private Repository)

What kind of app source is it?

SSH URL: git@github.com:GovReady/govready-apps-dev.git

The SSH URL to a private git repository, e.g. git@github.com:GovReady/govready-q.git.

Branch:

The name of the branch in the repository to read apps from. Leave blank to read from the

Path:

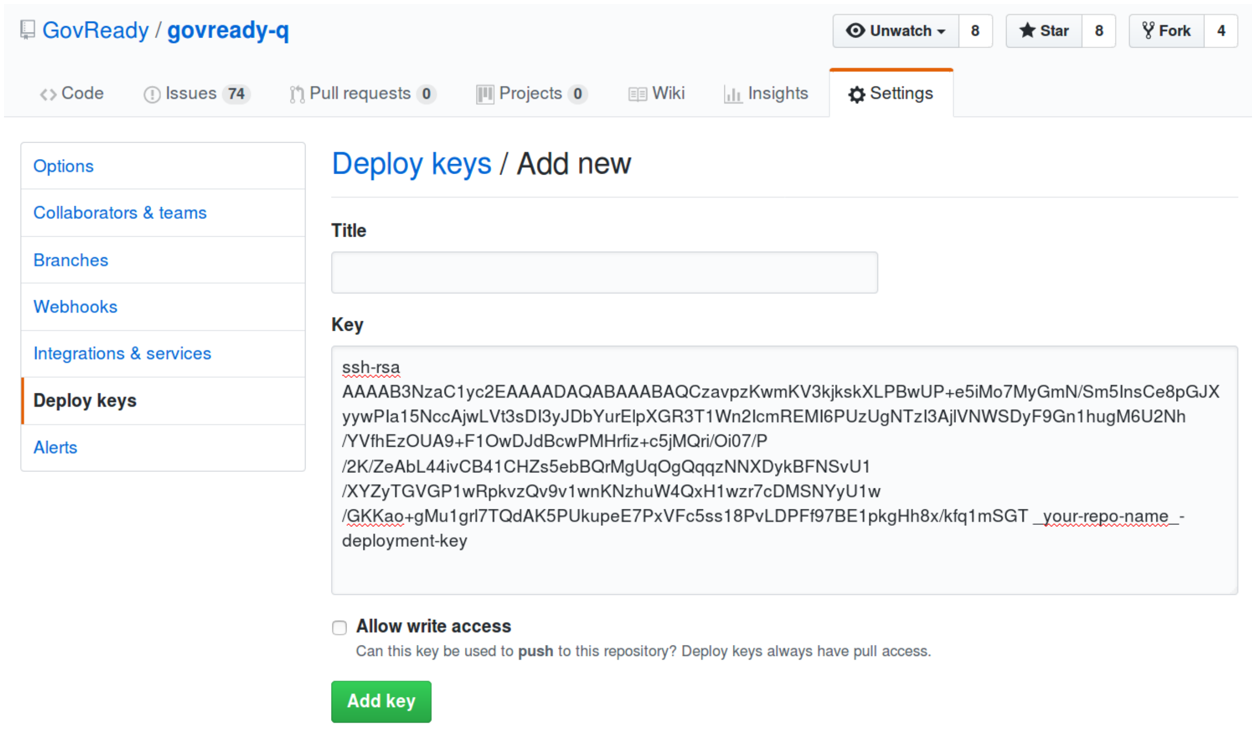
The path to the apps. For local directory AppSources, the local directory path. Otherwise th

SSH Key:
```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAs2r6cysJild5I5LJFyzwcFD/nuYjKO
zMhpjf0puSJ7AnvKRi
```

Paste an SSH private key here and send the public key to the repository owner. For GitHub

The other fields can be left blank. `Path` and `Branch` can be set the same as with the Git Repository over HTTPS source type (see above).

Copy the public key in the newly generated file `repo_deploy_key.pub` into the deploy keys section of your source code repository. Here is what that looks like on GitHub:



Deploy keys / Add new

Title

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCzavpzKwmKV3kjkskXLPBwUP+e5iMo7MyGmN/Sm5InsCe8pGJX
yywPIa15NccAjwLVt3sDI3yJDbYurElpXGR3T1Wn2IcmREMI6PUzUgNTzI3AjIVNWSDyF9Gn1hugM6U2Nh
/YVfhEzOUA9+F1OwDJdBcwPMHrfiz+c5jMQri/Oi07/P
/2K/ZeAbL44ivCB41CHZs5ebBQrMgUqOgQqqzNNXDykBFNSvU1
/XYZyTGVGP1wRpkvzQv9v1wnKNzhuW4QxH1wzr7cDMSNYyU1w
/GKKao+gMu1grl7TQdAK5PUkupeE7PxVFc5ss18PvLDPFf97BE1pkgHh8x/kfq1mSGT _your-repo-name_-
deployment-key
```

☐ Allow write access

Can this key be used to **push** to this repository? Deploy keys always have pull access.

Make the key read only by leaving "Allow write access" field unchecked and click `Add the key` to save the key.

GitHub Repository using the GitHub API

This source type can be used to access private GitHub repositories using a GitHub username and password or a username and personal access token.

Set the `Repository` field to the organization name and repository name, separated by a slash, as in the repository's URL following `github.com/`. In `Other Parameters`, paste a small YAML-formatted document holding a GitHub username and password or username and personal access token, formatted as follows:

```
auth:
  user: 'myusername'
  pw: 'mypassword'
```

The other fields can be left blank. `Branch` can be set the same as with the Git Repository over HTTPS source type (see above).

Since this source type requires user credentials, it should be avoided for production deployments in favor of using Git Repository over SSH.

4.3.3 Controlling access to apps

Controlling which organizations in a Q deployment can access which apps is done via the App Sources table.

The "Available to all" field of App Source, which is on by default, gives all users of all organizations the ability to start an app provided by the App Source.

If the "Available to all" field is unchecked, then only users within white-listed organizations can start apps provided by the App Source. The white-list is a multi-select box on the App Source page.

Removing access to a App Source does not affect any apps that have already been started by a user.

App executable content

Apps can contain executable content (some of which is disabled by default):

- JavaScript executed by the client browser contained within page HTML, via module template content.
- JavaScript executed by the client browser served as a static asset and referenced by a `<script>` tag.

Both sources of Javascript execute within the context of pages on the domain that the Q site itself runs on, which means the scripts have access to the page DOM, cookies, localStorage, etc. These scripts must only be enabled if they are trusted for these environments.

Javascript static assets (**but not Javascript in module templates** - this is a TODO) are therefore disabled by default. (Javascript static assets are disabled by serving them with an incorrect MIME type.)

To enable these scripts, the `Trust assets` flag must be true on the App Source that provides the app. This flag must only be true if any Apps provided by the App Source, including Apps already loaded into Q, are trusted to have executable content that may have as much client or server-side access as the Q instance does itself.

4.4 Modules, Questions, and Documents YAML Reference

A module and its questions are defined in YAML specification files. The schema for the specification files is as follows:

4.4.1 Module

Each file is a Module. A Module has the following required fields:

```
id: module_id
title: Your Title Here
```

The `module_id` must match the file name that the YAML is saved into, without the path or file extension.

Several optional fields can be specified:

```
type: project
version: 1
instance-name: "Module for {{q1}}"
invitation-message: "Can you tell me about {{question.text}} and let me know when you␣
↪are done?"
icon: app.png
```

The `type` field is set to `project` just when the Module is to be offered to users when they start a new Project. (`system-project` is used for project-like modules that are system controlled and not offered to the user.)

The `version` field is used only to force changes in the specification to be considered incompatible with any existing user answers (see Updating Modules).

The `instance-name` is a template to generate a dynamic title for in-progress and completed modules. The `instance-name` is rendered like other Module documents but it is always specified in `text` format (see Documents).

For modules that define the root of an application, `icon` specifies a static asset (in the `assets` directory) to use as an application icon.

In addition, a Module may have an `introduction` document (for projects, the introduction appears at the top of the project page; for other module types, it appears as an implicit initial interstitial) and a list of one or more `output` documents. For example:

```
introduction:
  format: markdown
  template: |
    Welcome to the module.

    This module should take you two minutes.

output:
  - title: Document 1
    format: markdown
    template: |
      # Document for {{project}}

      Hello! This is the output of the module. You answered {{q1}}.

  - title: Document 2
    glyphicon: dashboard
    format: html
    template: |
      <h1># Document for {{project}}</h1>

      <p>Hello! This is the output of the module. You answered {{q1}}.</p>
```

The format of documents are described in a later section.

Finally, a Module contains a list of one or more Questions:

```
questions:
  - id: q1
    title: Your Favorite Animal
    prompt: What's your favorite animal?
    type: text

  - id: q2
    title: What Kind of Animal Is It
    prompt: How would you classify this animal?
    type: choice
    choices:
      - key: pet
        text: common pet
        help: Is this animal a common pet?
      - key: wild
        text: wild animal
        help: Is this animal an undomesticated wild animal?
```

The schema for questions is documented in a later section.

Additional fields for projects

Question Fields

The questions of projects are displayed in a layout of tabs and groups within each tab pane. Each question that shows up on a project page should specify its tab and group name (which are also the display strings):

```
questions:
- id: howto_ssp
```

(continues on next page)

(continued from previous page)

```
  title: "SSP 101: What's a System Security Plan"
  type: module
  module-id: howto_ssp
  tab: How To
  group: Start Here
  icon: ssp.png
```

`icon` specifies a static asset (in the `assets` directory) to use as an icon for the question. If the question's type is `module` and it is answered and the answer is a Task that has a top-level `icon` field, then the answer's icon is used instead.

Instead of `tab` and `group`, `placement: action-buttons` can be used instead to show the question in an action bar above the tabs, rather than in tabs.

Output Document Fields

Output documents of a project module that have an `id` field are used in the following ways:

- They are displayed in the Related Controls page for the project. Add a `title` attribute to set the heading text above the document's content.
- They can be accessed from higher-level apps into which this app has been added. In a higher-level app, access the rendered HTML value of the output document as `{{question.output_documents.document_id}}`.

When `display: top` is set on an output document, it is rendered above the *Your Answers* section.

Test Answers

Projects can provide sets of exemplar answers for use in test scripts. e.g.:

```
tests:
  test1: # <-- test suite ID
    description: "Sample data."
    answers:
      q1:
        answers:  # <-- answers to sub-task's questions
          q1: desktop
          q2: My Secure Tool
```

4.4.2 Documents

Documents occur as `introduction` and `output` documents of Modules, and a restricted form of documents also occurs in Question prompts (see Questions below). A document appearing in the output documents list is given as:

```
output:
  - id: mydoc
    title: Document 1
    format: markdown
    template: |
      Hello!
```

The `id` and `title` fields are generally optional and are used for output documents only. An `id` is required to make the document downloadable. The fields also have special uses in projects (see above). The `format` field is described below.

The document can also be stored in a separate file by replacing the document data in the module YAML file with a filename and placing the document properties and template in the named file, as in:

```
# module.yaml
output:
  - mydoc.md

# mydoc.md
id: mydoc
title: Document 1
format: markdown
...
Hello!
```

When using a separate file, the document properties (`id`, `title`, and `format`) are given in a YAML block at the top of the file. A line containing just three dots signifies the end of the YAML block, separating it from the document template. The document template follows.

Document Format

The `introduction` and `output` documents of Modules allow a format to be specified. The document formats are:

- `markdown` — The document is entered in CommonMark (quick guide) in the specification file, but it will be rendered into a richly formatted presentation on screen.
- `html` — The document is given in raw HTML, but it will be rendered on screen.
- `text` — The document is given in plain text, and it will display as preformatted (fixed-width) text on screen.
- `json`, `yaml` — Experimental.

Additional Markdown Notes

Documents specified in `markdown` format are rendered according to the CommonMark 0.25 specification.

Note that for some things like tables, it is necessary to insert raw HTML right into the document, which is acceptable CommonMark. To create a table:

```
<table><thead><th>

Col 1

</th>
<th>

Col 2

</th>
</thead>
<tbody><tr><td>

Some [commonmark](http://www.google.com) within the cell.

</td>
<td>

More *content.*
```

(continues on next page)

(continued from previous page)

```
</td></tr></tbody></table>
```

Some of the newlines are necessary to get CommonMark to go out of raw HTML mode and back into parsing CommonMark.

Document Templating

All document formats are evaluated as Jinja2 templates. That means within your document you can embed special tags that are replaced prior to the document being displayed to the user:

- `{{ question_id }}` will be replaced with the user's answer to the question whose `id` is `question_id`. For choice-type questions, the value is replaced by the choice `key`. Use `{{ question_id.text }}` to get display text. See the question types documentation below for details.
- `{% if question_id == 'value' %}....{% endif %}` is a conditional block. The contents inside the block (`....`) will be included in the output if the condition is true. In this example, the contents inside the block will be included in the output if the user's answer to `question_id` is `value`.

Output documents and question prompts have access to the user's answers to questions in question variables. (The introduction document does not have access to the user's answers because questions have not yet been answered.)

The following information is also available within the output template for each question as of version `v0.8.6`:

- `{{ question_id.not_yet_answered }}` Question has not yet been answered.
- `{{ question_id.answered }}`Question has an answer either by user or was imputed, but not imputed `null` or answered `null`.
- `{{ question_id.imputed }}` Question considered "answered" but no TaskAnswerHistory record exists in the database for question meaning a user didn't provide the answer.
- `{{ question_id.skipped }}` Question has a null answer either because imputed `null` or the user skipped it.
- `{{ question_id.skipped_by_user }}` Question has a `null` answer because used a skip button (e.g., question wasn't imputed `null`).
- `{{ question_id.skipped_reason }}` Question's indicated reason for skipping (e.g. "I don't know" or "It doesn't apply")
- `{{ question_id.unsure }}` If question was answered by a user, its unsure flag. (NOTE: Purpose of this flag was to allow users to indicate uncertainty in the answer. Due to usability issues however, this feature is currently hidden.)
- `{{ question_id.date_answered }}` Question answered date.
- `{{ question_id.reviewed_state }}` Question reviewed value.

All documents also have access to the project title as `{{project}}`.

Project Documents

In addition to the `output` documents described above, a project module may also have a `snippet` that defines how a project appears in the project listing page:

```
snippet:
  format: markdown
  template: |
    Project {{name}}
```

4.4.3 Module Assets

Modules often make use of assets outside of the YAML file.

Static Assets

Static assets such as images can be referenced in module content (introductions, question prompts, and output documents). These assets are exposed by the Q web server in its static path. Place static assets in an `assets` subdirectory where the module is. When the asset is referenced in a Markdown document template, its path will be rewritten to be its public (virtual) path on the web server.

For example, to include an image in a module introduction add the image in the Markdown template:

```
module.yaml
-----------


...
format: markdown
template: |
  ![](my_image.png)
...
```

Place the module and image files at the path:

```
module.yaml
assets/my_image.png
```

Private Assets

Private assets are other files that are stored with a module but are not exposed by the web server. The directory provides a place to store files for internal use during module development.

Place private assets in a `private-assets` subdirectory next to the module YAML file.

4.4.4 Questions

Questions have the following required fields:

```
- id: q1
  title: Your Favorite Animal
  prompt: What's your favorite animal?
  type: text
```

The question `id` is used to refer to this Question in other questions and in the output documents.

The `title` is used to describe the Question in places where a long-form prompt would not be appropriate.

The prompt is the text the user is prompted with when presented with the question. The prompt is rendered like other Module documents but it is always specified in markdown format (see Documents). The first line (paragraph) of the prompt is shown in larger, bold type.

A question may have other optional fields that provide the user with other information, such as:

```
examples:
- example: |
   First example.
- example: |
   Second example.
reference_text: See NIST SP 800-171 page 102.
```

Like the prompt, each entry inside examples and the reference_text are Markdown templates.

Removing a question, changing a question type, and other changes as noted below are incompatible changes (see Updating Modules).

Question Types

text

This type asks the user for a single line of free-form text. The text cannot be empty.

A placeholder can be specified which places ghosted "placeholder" text inside the form field when the user has not yet entered anything. A default value can be specified, instead, which fills in the field with a value that the user can edit (or not) before submitting the answer. The placeholder and default fields are rendered like other Module documents — just like the prompt.

help text can be specified which provides an additional prompt smaller and below the field input.

Example:

```
- id: q1
  title: Your Favorite Animal
  prompt: What's your favorite animal?
  type: text
  placeholder: enter a type of animal
  help: Examples: dog, cat, turtle, lion
```

In document templates and impute conditions, the value of text questions is simply the text the user entered.

password

This type asks the user for a password. It is the same as the text question type, except that a password input field is used to mask the input. help can be specified. placeholder and default are not allowed.

email-address

This type asks the user for an email address. It is the same as the text question type, except that the value entered must be a valid email address. placeholder, default, and help can be specified.

url

This type asks the user for a web address (a URL). It is the same as the text question type, except that the value entered must be a valid web address. placeholder, default, and help can be specified. The web address is not checked for existence — only the form (syntax) of the address is checked.

longtext

This type asks the user for free-form text using a large rich text input area that allows for multiple lines of text and some simple formatting. The text cannot be empty.

A default value can be specified, which fills in the field with a value that the user can edit (or not) before submitting the answer. The field is rendered like other Module documents — just like the prompt. It is given in Markdown.

help text can be specified which provides an additional prompt smaller and below the field input.

In document templates and impute conditions, the value of longtext questions is the text the user entered, as a string, with rich formatted represented in CommonMark. In document templates, the text is automatically converted back to rich formatting.

date

This type asks the user for a date.

help text can be specified which provides an additional prompt smaller and below the field input.

In document templates and impute conditions, the value of date questions is a text string in YYYY-MM-DD format.

choice

This type asks the user to choose one of several options. The options are given as:

```
choices:
  - key: pet
    text: common pet
    help: Is this animal a common pet?
  - key: wild
    text: wild animal
    help: Is this animal an undomesticated wild animal?
```

The user must select exactly one choice.

The help text is optional. It is displayed smaller and below each choice. (Unlike some other question types, there is no help field on the question as a whole.)

In document templates and impute conditions, the value of choice questions is the key of the choice selected by the user. Use questionid.text to access the display text for the choice.

Removing a choice is an incompatible change (see Updating Modules).

yesno

This type is the same as choice but with built-in choices for yes and no. It is the same as a choice question type with these choices:

```
choices:
  - key: yes
    text: Yes
  - key: no
    text: No
```

The user *must* choose either yes or no.

`multiple-choice`

The `multiple-choice` question type is similar to the `choice` question type except that:

- The user can select multiple choices.
- In document templates and impute conditions, the value of `multiple-choice` questions is a list of the `keys` of the choices selected by the user. When used bare, this renders as a comma-separated list of keys. One can use the *‘‘|length‘* filter <http://jinja.pocoo.org/docs/dev/templates/#length>‘_ and `{% for ... in ... %}... {% endfor %}` loops to access the individual choices the user selected. Use `questionid.text` to render a comma-separated list of the display text of the selected choices.
- `min` and `max` may be specified. If `min` is specified, it must be greater than or equal to zero and requires that the user choose at least that many choices. If `max` is specified, it must be greater than or equal to one (and if `min` is specified, it must be at least `min`) and requires that the user choose at most that number of choices.

Increasing the `min` or decreasing the `max` are incompatible changes (see Updating Modules).

`integer`

This question type asks for a numeric, integer input.

If `min` and `max` are set, then the value is restricted to that range. If `min` is omitted, then negative numbers are allowed!

As with the text question types, `placeholder` and `help` text can also be specified.

In document templates and impute conditions, the value of `integer` questions is the numeric value entered by the user.

`real`

This question type asks for a numeric input, allowing for real (floating-point) numbers.

If `min` and `max` are set, then the value is restricted to that range. If `min` is omitted, then negative numbers are allowed!

As with the text question types, `placeholder` and `help` text can also be specified and in document templates and impute conditions the value of these questions is the numeric value entered by the user. .

`file`

This question type asks the user to upload a file.

`help` text can also be specified, as in the text question types.

By default, any type of file is permitted to be uploaded. If the optional `file-type` field is set, the uploaded file is validated to be of a particular type. Supported values for the `file-type` field are:

- `image`: Ensures the file is an image. The uploaded file is converted to PNG format internally.

If `file-type` is `image`, then some image transformation can be run, e.g.:

```
- id: logo
  title: Logo
  prompt: Upload a logo.
  type: file
  file-type: image
  image:
    max-size:
      width: 60
      height: 60
```

If `image->max-size` is given, then the image will be resized prior to being saved internally so that its width and height do not exceed the given dimensions.

In document templates and impute conditions, the value of these questions is a Python dict (JSON object) containing `url` (a download URL) and `size` (in bytes) fields.

`module`, `module-set`

These question type prompt the user to select another completed module as the answer to the question. The `module-id` field specifies the ID of another module specification. The `module` question type allows for a single other module to answer the question. The `module-set` question type allows for zero or more other modules to answer the question.

The `module-id` field specifies a module ID as it occurs in the `id` field of another YAML file in the same application.

Example

Here's an example of the `module` question type:

```
- id: evidence
  title: Evidence
  type: module
  module-id: evidence
  prompt: |
    Provide evidence of your properly configured firewall, if possible.
  impute:
    - condition: not(have_other_dmz == 'ad_hoc_dmz')
      value: ~
```

App protocols

Instead of using `module-id`, a `protocol` can be specified instead. A protocol is a globally unique identifier that apps in the Compliance Store use to indicate that their questions and output documents meet a certain criteria (i.e. implement the protocol). When a user attempts to answer a `module` or `module-set` question that uses `protocol` instead of `module-id`, instead of starting a particular named module, the user instead can start any app from the Compliance Store that implements the protocol.

For example:

```
- id: evidence
  title: Evidence
```

(continues on next page)

(continued from previous page)

```
  type: module
  protocol: govready.com/apps/compliance/2017/nist-sp-800-171-r1-ssp
  prompt: |
    Provide evidence of your properly configured firewall, if possible.
```

When a user answers this question, they will be redirected to the Compliance Store but will be offered only apps that implement the protocol `govready.com/apps/compliance/2017/nist-sp-800-171-r1-ssp`.

An app implements a protocol by having a `protocol:` field at the top level of the app's YAML specification file with the same value. For instance, the following app would be offered in the Compliance Store for this example question:

```
id: app
title: My App
type: project
protocol: govready.com/apps/compliance/2017/nist-sp-800-171-r1-ssp
```

Both protocol fields can be either a single string or a list of strings. When the question `protocol` value is a list, then only apps which implement all of the listed protocols will be offered.

Question type details

Changing the `module-id` or `protocol` is considered an incompatible change (see Updating Modules), and if the referenced Module's specification is changed on disk in an incompatible way with existing user answers, the Module in which the question occurs is also considered to have an incompatible change. Thus an incompatible change in a module triggers an incompatible change in any other Module that refers to it (and so on recursively).

In document templates and impute conditions, the value of `module` questions is a dictionary of the answers to that module. For example, if `q5` is the ID of a question whose type is `module`, then `{{q5.q1}}` will provide the answer to `q1` within the module the user selected that answers `q5`.

`interstitial`

An `interstitial` question is not really a question at all! The `prompt` contains template content, as with other questions, but it is typically longer content with deeper explanatory text. The user is not asked to enter any information.

In document templates and impute conditions, the value of `interstitial` questions is always a null value.

`raw`

This type is meant for questions that are always imputed (i.e. that are never presented to the user) and where the answer value can be any JSON-serializable Python data structure, as given by the impute value (see Imputing Answers below).

This question type should be avoided if one of the other question types specifies a more narrow data type. For instance, if the imputed value is always a string, the `text` or `longtext` question types should be used instead.

Imputing Answers

The answer to one question may provide the answer to another. In such cases, the latter question is said to have an imputed value and the user is not asked to answer the question. To impute a value, specify on the question whose value is being imputed:

```
impute:
  - condition: q1 == 'no'
    value: don't know
```

This example says that if the answer to q1 is no, then the answer to this question is don't know.

The condition is a Jinja2 expression. Any question can be referred to in the expression (by its id). Questions are tested on their internal values. For choice and multiple-choice questions, their values are their keys, not their label text, and multiple-choice questions are *lists* of keys. If condition is omitted, the imputed value is always taken (i.e. the condition is implicitly met).

The value provided must be a valid value for the question type it is a part of. For choice questions, the value must be a choice key, not the label text. For multiple-choice questions, the value must be a *list* of keys.

Multiple condition/value blocks can be provided. They are evaluated in order, with the first matching condition taking precedence.

```
impute:
  - condition: q1 == 'no'
    value: I don't know.
  - condition: q1 == 'yes'
    value: I do know.
```

The value field can be evaluated as a Jinja2 expression, just like the condition, if value-mode is set to expression. This can be used to pull forward the answers of previous questions:

```
impute:
  - condition: q1 == 'same-as-q0'
    value: q0
    value-mode: expression
```

value-mode can also be template to evaluate the value as a Jinja2 template, which will yield a text value.

In both conditions and expression-type values, as well as in documents, the variables you can use are:

- ids of questions in the module
- question_id.subquestion_id to access questions within the tasks that are assigned as answers to module-type questions
- project, which gives the project name
- project.question_id, project.question_id.subquestion_id, etc. to access questions within the project
- organization, which gives the organization name

We also have a function to retrieve the URL of a module's static assets, e.g.:

```
<script src="{{static_asset_path_for('myscript.js')}}"></script>
```

4.4.5 Question Order

The order in which Questions are asked is determined through an algorithm. The algorithm determines which questions need to be asked before other questions and which need to be asked in order to generate the output documents.

The only Questions that are asked of the user are those that are mentioned in any of the output templates or other Questions that required to be asked before those mentioned Questions can be answered.

If a Question mentions another question in its prompt text or impute conditions, the other question must be answered first. A Question can also list other Questions that should be answered first as:

```
ask-first:
 - q1
 - q2
```

4.4.6 Updating Modules

When a Module file specification is changed, the change is considered "compatible" or "incompatible" with existing user answers.

Many changes are "compatible": Changing the introduction or output documents, question prompts, and adding new questions and choices are all compatible changes. These changes can be made "live" on any existing user answers.

Other changes are "incompatible": Removing a choice is an incompatible change because a user may have already chosen it. Removing a question is incompatible because it would result in a loss of user data.

When there is an incompatible change in a Module specification, a new iteration of the Module will be stored in the program database but existing user answers will continue to be tied to the previous iteration of the Module specification.

CHAPTER

FIVE

AUTOMATION API

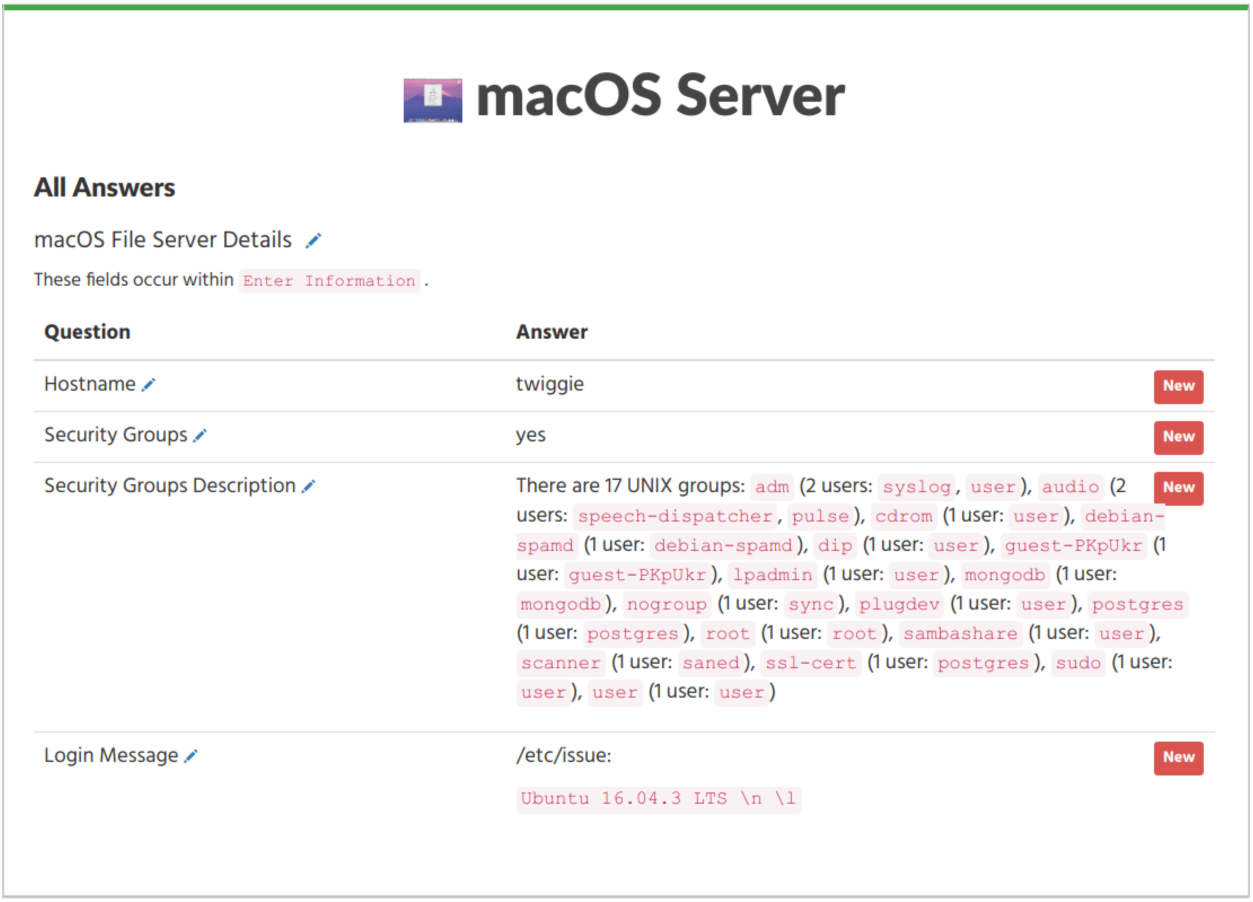
GovReady Q Compliance Apps can be updated with information gathered from live systems via the GovReady Q API. The benefit of this capability is that the documentation produced by GovReady Compliance Apps, such as System Security Plans (SSPs), can be assembled and updated with actual system data in an automated way.

5.1 Overview of the GovReady Q API

The GovReady Q API provides read and write access to the information stored in GovReady Q's question-and-answer data model. It is a RESTful API using HTTP GET and POST requests, API keys that are issued per user, and JSON for request and response data.

Each GovReady Compliance App provides a separate API, and each app's API is composed of fields for the same information the app would ask an end-user using the Q website in a web browser. The app's definition of questions to ask the end user (see Modules, Questions, and Documents) also define the data model of the API.

As an example, the screenshot below shows a demonstration of a macOS File Server compliance app. The app asks questions about the hostname of the server and the use of security groups.



macOS Server

All Answers

macOS File Server Details

These fields occur within `Enter Information`.

Question	Answer	
Hostname	twiggie	New
Security Groups	yes	New
Security Groups Description	There are 17 UNIX groups: `adm` (2 users: `syslog`, `user`), `audio` (2 users: `speech-dispatcher`, `pulse`), `cdrom` (1 user: `user`), `debian-spamd` (1 user: `debian-spamd`), `dip` (1 user: `user`), `guest-PKpUkr` (1 user: `guest-PKpUkr`), `lpadmin` (1 user: `user`), `mongodb` (1 user: `mongodb`), `nogroup` (1 user: `sync`), `plugdev` (1 user: `user`), `postgres` (1 user: `postgres`), `root` (1 user: `root`), `sambashare` (1 user: `user`), `scanner` (1 user: `saned`), `ssl-cert` (1 user: `postgres`), `sudo` (1 user: `user`), `user` (1 user: `user`)	New
Login Message	/etc/issue: `Ubuntu 16.04.3 LTS \n \l`	New

The answers to these questions can be both read from Q and written to Q using a JSON data structure:

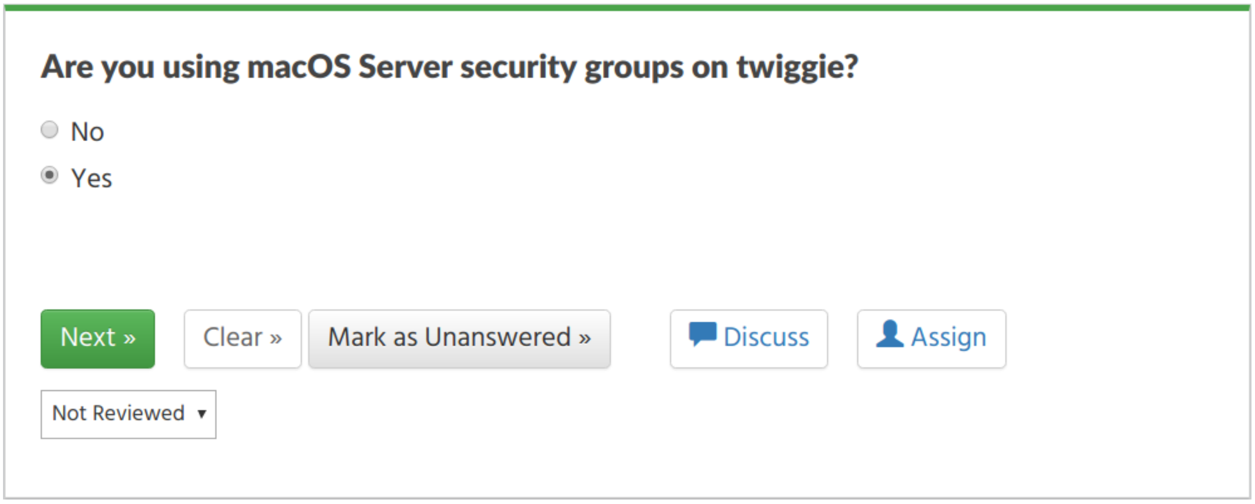
```
{
  "schema": "GovReady Q Project API 1.0",
  "project": {
    "file_server": {
      "hostname": "twiggie",

      "login_message": "/etc/issue:\n\n```Ubuntu 16.04.3 LTS \\n \\l```",
      "login_message.html": "<p>/etc/issue:</p>\n<p><code>Ubuntu 16.04.3 LTS
           \\n \\l</code></p>",

      "using_security_groups": "yes",
      "using_security_groups.text": "Yes"

      "security_groups_description": "There are 17 UNIX groups: `adm` (2 users:
           `syslog`, `user`), `audio` (2 users: `speech-dispatcher`, `pulse`), ...",
      "security_groups_description.html": "<p>There are 17 UNIX groups: <code>
           adm</code> (2 users: <code>syslog</code>, <code>user</code>), ..."
    }
  }
}
```

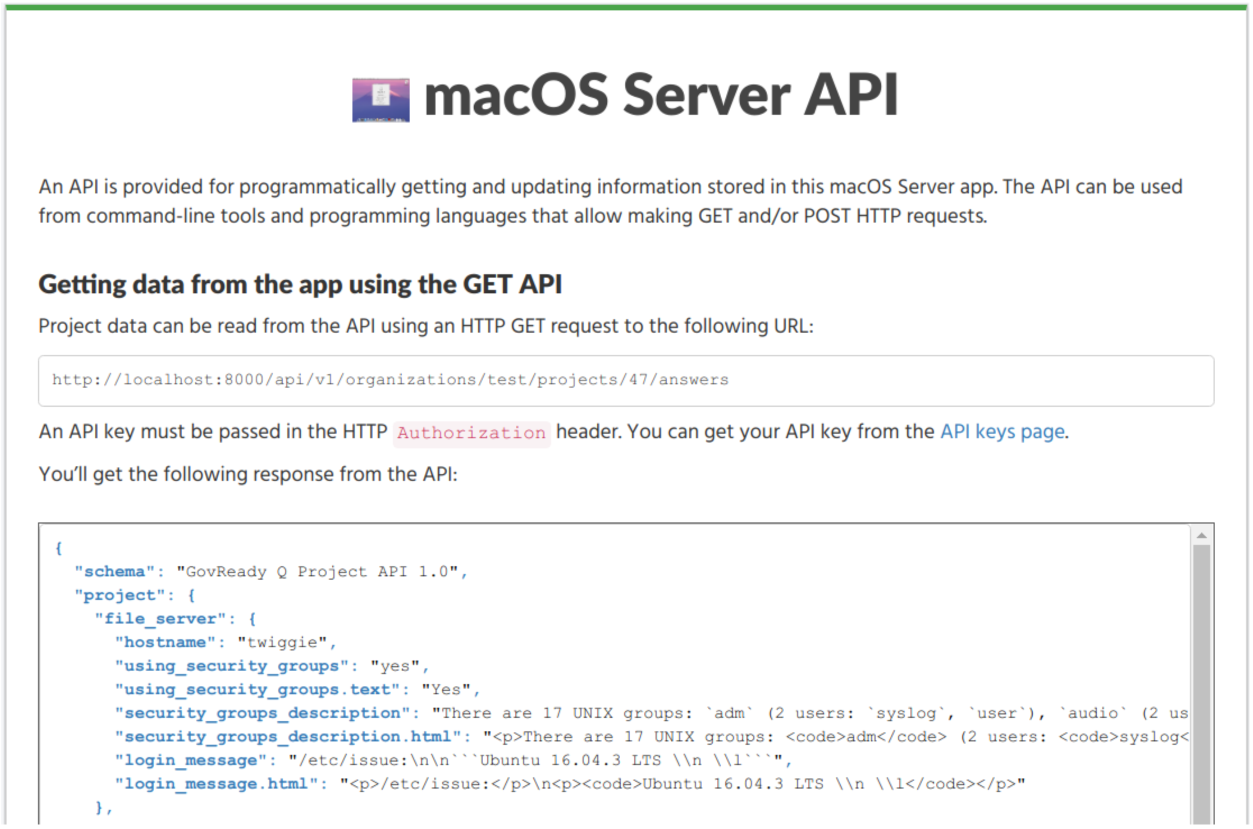
The second question, shown below, is a yes-no question. In the web browser this question appears as a radio select question with Yes and No choices. In the JSON data structure, shown above, it is encoded as the JSON strings `"yes"` or `"no"`.



Are you using macOS Server security groups on twiggie?

No [✓ Yes]

Each compliance app that has been started in GovReady Q and added to a project folder provides an `API Docs` page with samples and data schema documentation:



macOS Server API

An API is provided for programmatically getting and updating information stored in this macOS Server app. The API can be used from command-line tools and programming languages that allow making GET and/or POST HTTP requests.

Getting data from the app using the GET API

Project data can be read from the API using an HTTP GET request to the following URL:

```
http://localhost:8000/api/v1/organizations/test/projects/47/answers
```

An API key must be passed in the HTTP `Authorization` header. You can get your API key from the API keys page.

You'll get the following response from the API:

```
{
  "schema": "GovReady Q Project API 1.0",
  "project": {
    "file_server": {
      "hostname": "twiggie",
      "using_security_groups": "yes",
      "using_security_groups.text": "Yes",
      "security_groups_description": "There are 17 UNIX groups: `adm` (2 users: `syslog`, `user`), `audio` (2 us
      "security_groups_description.html": "<p>There are 17 UNIX groups: <code>adm</code> (2 users: <code>syslog<
      "login_message": "/etc/issue:\n\n```Ubuntu 16.04.3 LTS \\n \\l```",
      "login_message.html": "<p>/etc/issue:</p>\n<p><code>Ubuntu 16.04.3 LTS \\n \\l</code></p>"
    },
```

5.2 Using the Compliance API

To use a compliance app API, an app must already be started in GovReady Q by adding it to a project folder **and** its modules must be started (but need not be completed) for them to be accessible from the API.

5.2.1 API keys

Every call to the API requires an API key. Each user has three API keys listed on their API Keys page, which can be found in the user drop-down menu on Q: a read-only API key, a read-write API key, and a write-only API key:

- The read-only API key gives external tools the ability to see all data values that the associated user can see on Q, but the API key cannot be used to change any data values.
- The read-write API key gives external tools the ability to see and make changes to anything the associated user can see and make changes to on Q.
- The write-only API key gives external tools the ability to make changes to anything the associated user can make changes to on Q but does not include the ability to see any data values stored in Q. The write-only key is useful in situations where the external tool needs to be able to upload data but does not need to read existing data values.

5.2.2 Getting data from the app using the GET API

Project data can be read from the API using an HTTP GET request to a URL of the following pattern:

`{site base URL}/api/v1/projects/{project id}/answers`

The complete URL can be found on the `API Docs` page for a compliance app that has been started and added to a project folder.

An API key must be passed in the HTTP `Authorization` header. You can get your API key from the API Keys page, which is found in the site header menu in the user drop-down.

The API response is a JSON data structure similar to the example above. The schema of the response object is documented on the app's `API Docs` page. Further information can also be found below.

If you are using an operating system with a command line and the `curl` tool, you can try out the API by running:

```
curl --header "Authorization: {your-api-key}" \
  {site base URL}/api/v1/projects/{project id}/answers
```

5.2.3 Updating data using a POST request with form data

There are two types of POST requests that can be used to update app data. In the first type, described in this section, data values are provided as key-value pairs using the regular web browser form submission method. (In the second form, described below, answers are provided using a JSON data structure that is formatted the same as the JSON data structure returned by a GET request, which may be more appropriate when submitting non-textual and non-binary content.)

In each of the key-value pairs submitted in the POST request, the *key* is a dotted-path question ID. The key always begins with `project.` and is followed by the property names on the path to the question being updated, according to the JSON data structure, with property names separated by the `.` character.

The *value* of each key-value pair is an answer submitted either as plain text or, for file-type questions, as a binary file. If submitted as plain text and the question expects non-text data, such as a number, the value will be converted. When uploading a binary file, the multipart/form-data content type must be used for the POST request.

As with the GET API, an API key must be passed in the HTTP header. An API key with write permission must be used. You can get an API key from the API keys page on your Q site.

If you are using an operating system with a command line and the `curl` tool, you can try out the API by running:

```
curl \
  --header "Authorization: <i>your-api-key</i>" \
  -F project.question.subquestion1=datavalue \
  -F project.question.subquestion2=datavalue \
  {site base URL}/api/v1/projects/{project id}/answers
```

For a file upload, use `-F @filename.ext`. `curl`&rsquo;s `-d` option can be used in place of `-F` if none of the fields are file uploads.

5.2.4 Updating data using a POST request with JSON

Use a POST request instead a GET request to the same URL to update data stored in the app. Data values to save in the app are included in the request body as JSON in the same format as returned by the GET request.

The POST request body always includes:

```
{
  "schema": "GovReady Q Project API 1.0",
  "project": {
    ...
  }
}
```

Answer data is placed inside the `project` field.

As with the GET API, an API key must be passed in the HTTP `Authorization` header. An API key with write permission must be used. You can get your API key from the API Keys page, which is found in the site header menu in the user drop-down.

If you are using an operating system with a command line and the `curl` tool, you can try out the API by placing the JSON request data in a file named `data.json` and then running:

```
curl --header "Authorization: {your-api-key}" \
  -XPOST --data @data.json --header "Content-Type: application/json" \
  {site base URL}/api/v1/projects/{project id}/answers
```

5.3 API Data Schema

Each compliance app documents its data schema on its API Docs page, which can be found inside the compliance app after it has been started and added to a project folder.

Each question defined by the app — which it would ask an end-user when in a web browser — is exposed as a field in the JSON data structure. The field types are:

- Text, password, email-address, and URL fields: Encoded as a JSON string. Email-address fields must contain valid email addresses. URL fields must contain valid URLs.
- Long text fields, which hold multi-paragraph text: Encoded as a JSON string with formatting expressed in CommonMark (i.e. Markdown).
- Date fields: Encoded as a JSON string in YYYY-MM-DD format.
- Single-choice and yes-no fields: Encoded as a JSON string holding a programmatic identifier for the selected choice. Yes-no fields use the identifiers `yes` and `no`.

- Multiple-choice fields: Encoded as a JSON array of strings, where each string is a programmatic identifier for a selected choice.
- Integer and real number fields: Encoded as a JSON number. Integer fields must contain integer values.
- File fields: Encoded as a JSON object containing the properties `url` (a link to download the file content), `type` (the MIME type), and `size` (the size of the file in bytes).
- "Module" questions create recursive structures and are encoded as JSON objects. "Module-set" questions are encoded as JSON arrays of JSON objects.

All fields can also hold `null`, which indicates the question has been explicitly "skipped." If a question is unanswered, it does not appear in the API.

Single-choice, multiple-choice, and yes-no fields also appear in human-readable form as a second read-only field that uses a `.text` suffix in the field's name. Long text fields have an HTML display form, in which the CommonMark is pre-rendered, in a parallel field with a `.html` suffix in the field's name. These fields cannot be used in the POST API.

More information about Q's data types can be found in Modules, Questions, and Documents.

CHAPTER

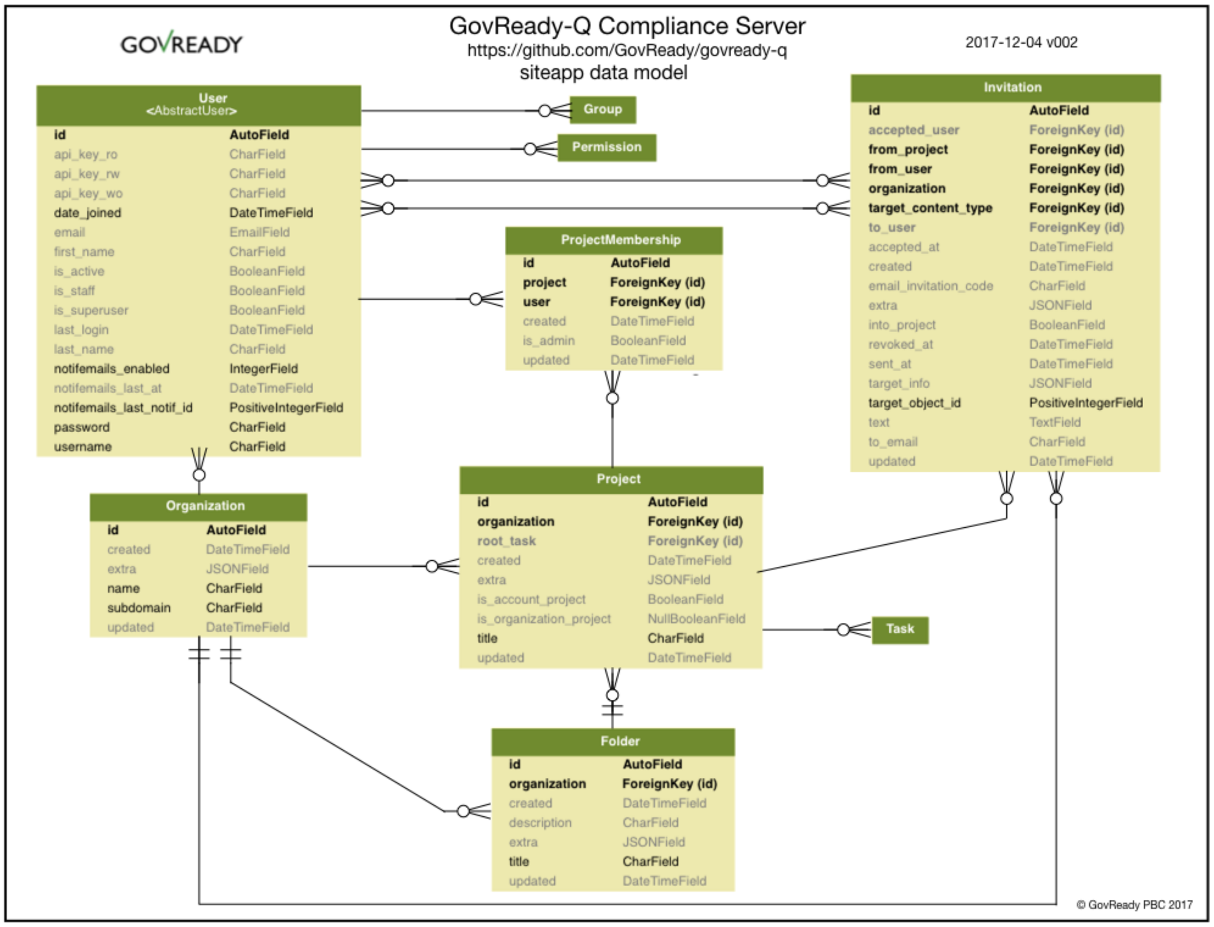
SIX

DATA DESIGN GUIDE

The documents in this section describe GovReady-Q's database design.

6.1 Users, Organizations, Projects, Folders, and Invitations

The diagram below provides a summary representation of GovReady-Q's Django `siteapp` data model, which handles users, organizations, projects and folders, and invitations.



The siteapp data model represents users who have membership in different projects. Users must be invited to projects. Projects belong to organizations.

Access control is based on organization and projects. Information cannot be shared across organizations and only limited information can be shared across projects within an organization.

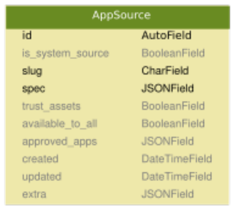
6.2 Compliance Apps, Modules, Questions, Tasks, and Answers

GovReady-Q is a governance, risk, and compliance platform for creating automated compliance processes ranging from gathering information from persons and computers to generating compliance artifacts.

Information gathering is at the heart of GovReady-Q's `guidedmodules` data model, which handles compliance apps, modules, questions, tasks, and answers. Eight database tables make up the `guidedmodules` data model. The complete data model spans three categories:

1. Compliance apps, which are reusable packages of questions, business logic, and document templates, are defined by the database tables `AppSource`, `AppVersion`, `Module`, `ModuleQuestion`, and `ModuleAsset`. Many compliance apps, and different versions of the same compliance app, can be in use simultaneously.
2. Information submitted by end-users to answer compliance app questions, as well as information submitted through the GovReady-Q API, is stored in the database tables `Task`, `TaskAnswer`, and `TaskAnswerHistory`.
3. Imputed answers and documents generated by the business logic stored in compliance apps, which are computed on-the-fly.

The tables are described is additional detail below and their relationships are summarized in the following diagram:



AppSource	
id	AutoField
is_system_source	BooleanField
slug	CharField
spec	JSONField
trust_assets	BooleanField
available_to_all	BooleanField
approved_apps	JSONField
created	DateTimeField
updated	DateTimeField
extra	JSONField

6.2.1 Compliance Apps (Questions, Business Logic, and Templates)

The smallest unit of a compliance app is a `Question`. Questions come in different types, such as text, number, and date (full list). Questions are grouped into questionnaires called `Modules`. And Modules are grouped into `AppVersion`, which are the versions of a compliance app that are loaded into the GovReady-Q database. AppVersions are loaded from AppSources, which define how to load compliance apps from remote sources such as GitHub or an on-premesis enterprise source control system.

`AppVersion`, `Modules`, and `Questions` define the structure of a compliance app but do not store any user-submitted content. Separating structure from content is a common pattern in application design and is motivated by several GovReady-Q goals:

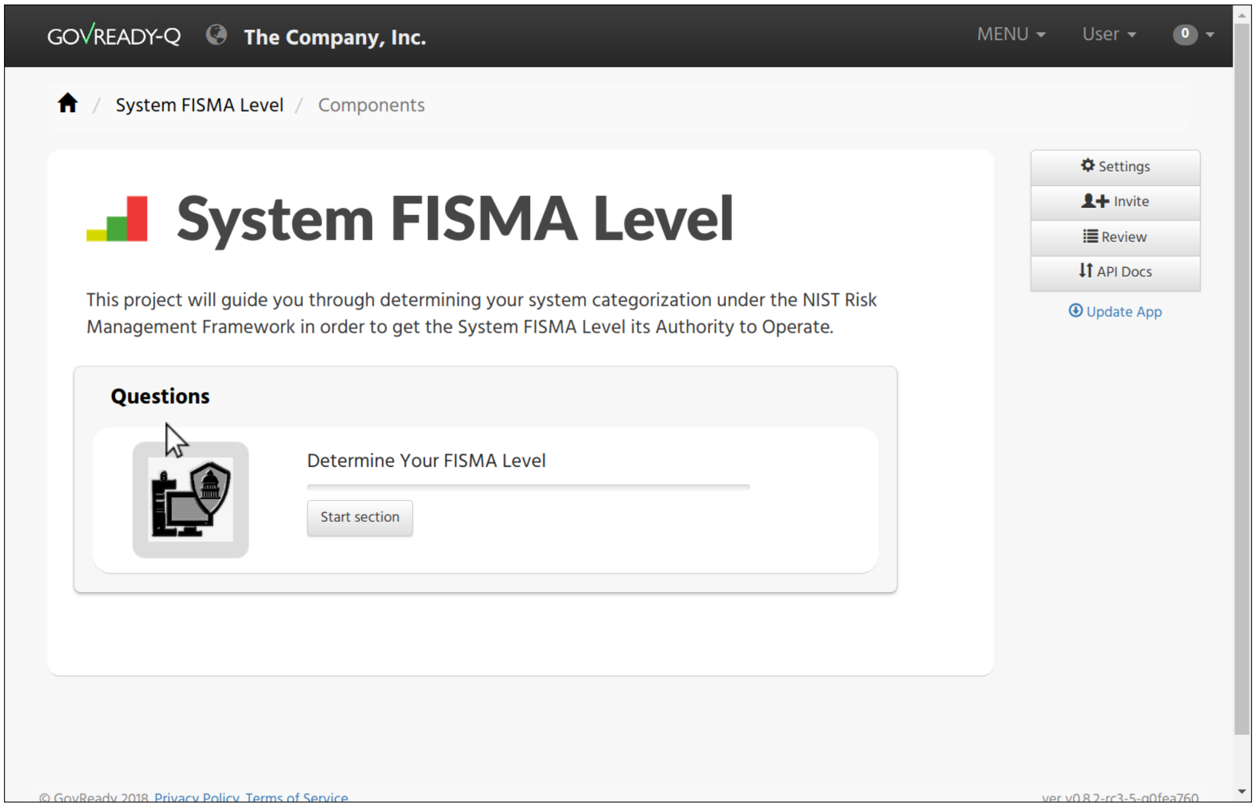
- Compliance apps are reusable and can be easily loaded into different installs of GovReady-Q at different organizations.
- Anyone can author compliance apps and they can be kept private or shared publicly.
- One type of Question allows the user to choose a complete set of answers to another Module, which allows question answers (i.e. user data) to be accessed from the business logic and templates of not only the Modules the Questions are defined in but also from other Modules and even other compliance apps.
- Compliance apps are versioned and apps that have been started by users can be updated in non-destructive ways, preserving answered questionnaires over the course of years,

GovReady-Q administrators will configure GovReady-Q with an `AppSource` record for each source of compliance apps that will be made available to GovReady-Q users. Each AppSource has a "slug," which is its unique name on the GovReady-Q installation. An AppSource also defines a remote location — such as a GitHub repository, local directory, or on-premise git server — to query for compliance apps, which will be listed in the app catalog. A typical GovReady-Q installation might have one AppSource providing compliance apps published by GovReady PBC and a second AppSource for providing compliance apps defined by the organization. Each AppSource defines the remote location as well as local permissions such as which compliance apps in the store to make available to GovReady-Q users and Organizations. See App Sources for more information.

Each time a compliance app is started by a user, an `AppSource` is queried for the latest version of the selected compliance app and an `AppVersion` is created that holds the complete set of questions, business logic, and templates defined in that version of the compliance app. Therefore there will be a separate AppVersion in the database for every version of every compliance app being used by GovReady-Q users.

Each AppVersion links back to the AppSource it was created from (the "source" field) and holds the name it was given (the "appname" field). Each AppVersion brings along with it the `Modules` and `Questions` defined in that version of the compliance app. In other words, Modules and Questions are specific to a AppVersion. Therefore if two versions of a compliance app are present in the database, a question that exists in both versions of the app is represented as two Question records — one for each version of the app. Similarly, there will be (at least) two Modules, one for each version of the app.

All compliance apps have at least two `Modules`. The first module, whose "module_name" identifier is always `app`, defines the layout of the starting page of a compliance app, which can list one or more `Modules` to complete:


GOVREADY-Q
The Company, Inc.
MENU
User
0
System FISMA Level / Components
Settings
Invite
Review
API Docs
Update App

System FISMA Level

This project will guide you through determining your system categorization under the NIST Risk Management Framework in order to get the System FISMA Level its Authority to Operate.

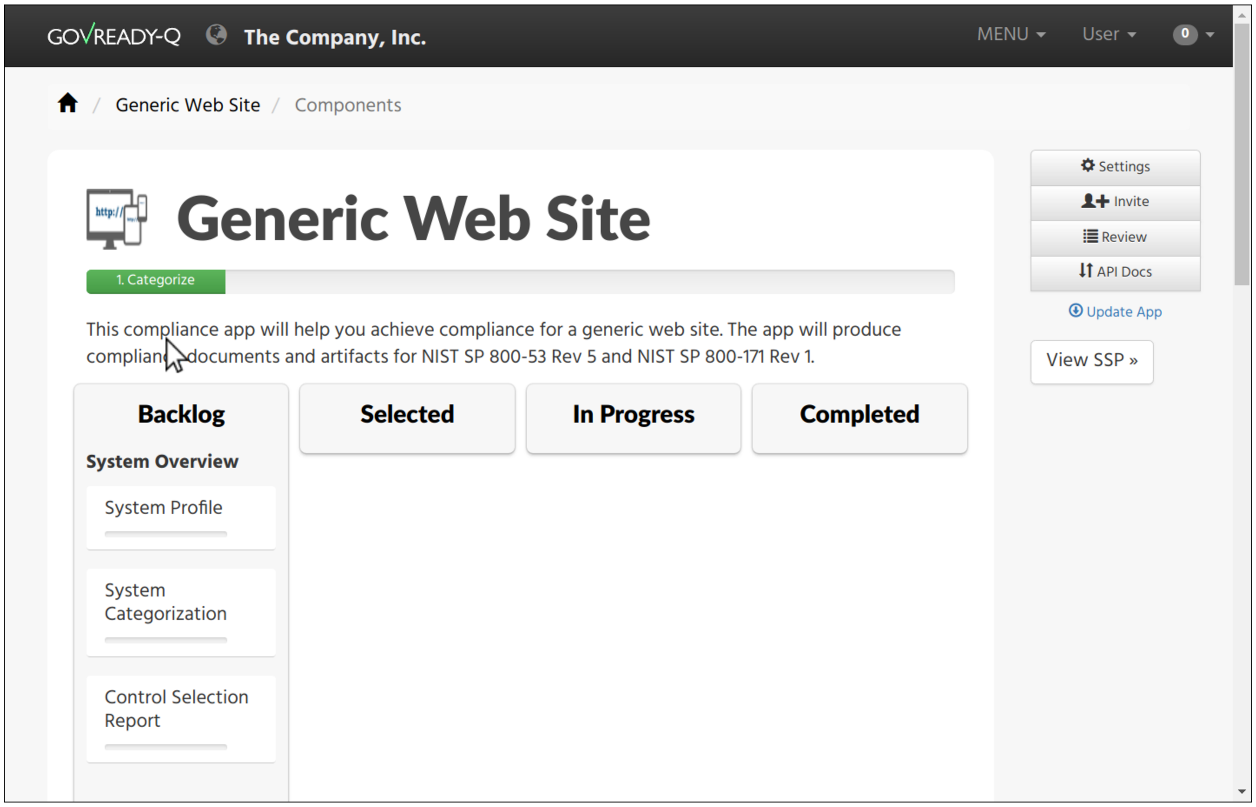
Questions

Determine Your FISMA Level

Start section

© GovReady 2018 Privacy Policy Terms of Service
ver v0.8.2-rc3-5-g0fea760

or a collection of modules and slots to start other compliance apps:



Generic Web Site

1. Categorize

This compliance app will help you achieve compliance for a generic web site. The app will produce compliance documents and artifacts for NIST SP 800-53 Rev 5 and NIST SP 800-171 Rev 1.

Backlog | Selected | In Progress | Completed

System Overview

System Profile

System Categorization

Control Selection Report

In both cases, the "cards" that represent modules to answer or slots for compliance apps to start are defined by

`Questions` in the "app" `Module`. When a user starts a module or selects a compliance app, that is recorded in the database as *answering* the respective Question in the "app" Module (more on that below).

Besides listing questions, `Modules` also define zero or more output documents. Each output document is generated by combining a template stored in the Module with user answers.

Similar to Modules, `ModuleQuestions` have a "key" field that uniquely identifies them within the Module they are defined in. ModuleQuestions store the question type (text, date, etc.), the prompt shown to users, impute conditions (see below), and other metadata.

`ModuleAssets` store a compliance app's static assets used by the app's templates. These assets often appear as images or other embedded media in output documents generated by the compliance app.

6.2.2 User Answers (Tasks and Answers)

When a user is completing the questions in a compliance app, their answers are stored in a separate set of database tables distinct from the tables used to store compliance app questions, business logic, and templates. The tables that hold answers are `Task`, `TaskAnwser`, and `TaskAnswerHistory`.

`Task` and `TaskAnswer` are parallel tables to `Module` and `ModuleQuestion` and are related to where user answers are stored. A `Task` is the instantiation of a `Module` that a GovReady-Q user or set of users are completing. A `TaskAnswer` is the instantiation of a `ModuleQuestion` that a GovReady-Q user has answered. All of the Tasks instantiated together for the same compliance app are related through the "project" field.

`TaskAnswerHistory` stores the complete history of user answers related to a TaskAnswer, i.e. to an instantiated question. The *current* answer to a question and its associated metadata are stored in the most recent TaskAnswerHistory record for a particular TaskAnswer (the one with the highest "id" value — "id"s are assigned to answers in strictly increasing order). Therefore only the most recent TaskAnswerHistory record for a TaskAnswer holds a current answer, and earlier TaskAnswerHistory records are for audit logging and tracking changes.

TaskAnswerHistory records have a "stored_value" field which holds the user's answer encoded in JSON, other metadata such as "answered_by" for which user provided the answer, "skipped_reason" and "unsure" which are flags set if the user skipped the question or wants to return to it later, and "reviewed" which holds workflow review state (e.g. if a reviewer marks the answer as approved).

This data model supports GovReady-Q design goals, such as:

- Compliance app modules and questions can be assigned to different users to answer.
- The answer to questions may change while a complete history of answers are preserved in an immutable record, including preserving past answer metadata such as who answered the question and whether the answer was approved by a reviewer.
- Answers are strongly typed: text, numbers, dates, choices, and so on are encoded in a JSON representation that preserves their data type.
- All questions can be skipped by storing `null` in "stored_value."

6.2.3 Imputed Answers and Output Documents

Compliance apps hold business logic and templates that are used to "impute" answers to questions and generate output documents, respectively. These computational outputs are not stored in the database. Instead, they are computed on-the-fly by GovReady-Q as they are needed, and the results of the computations are cached so long as they remain valid.

Imputed Answers

Imputation uses business logic rules to infer the answer to questions based on previous answers to questions. Imputation is used for a variety of purposes, such as:

- Hiding questions that are not applicable based on the answers to previous questions, by imputing `null` as the answer to the question.
- Pre-answering questions when the answer is known based on the answers to previous questions.
- Running business logic computations, such as computing a grade or gap analysis, and storing the result of the computation as the answer to the imputed question.

Questions whose answers are imputed are not asked of the user — the user may never see these questions at all. Some questions are designed to always be imputed to support the execution of business logic rules.

The results of imputation are not stored in the database because they are computed on-the-fly to ensure that the GovReady-Q always runs the business logic rules on the most recent, current set of answers to the questions. As a result, there may be no `TaskAnswer` or `TaskAnswerHistory` records for questions that have been imputed.

In certain circumstances, a question's answer may be imputed after a user already provided an answer to the question. In such cases, the user's answer remains in the database and appears in the database as the current answer to the question. However, when visiting GovReady-Q, imputed answers supersede user answers and only the imputed value will be used.

Output Documents

Compliance apps produce output documents. Each output document is generated by combining a template with the answers to questions (both user-inputted and imputed). Templates are typically written using Markdown syntax and are displayed in GovReady-Q as HTML documents, but they typically can also be downloaded in other formats such as a Microsoft Word document or PDF.

As with imputation, output documents are generated on-the-fly when they are viewed by GovReady-Q users. The generated documents are not stored in the database because they are computed on-the-fly to ensure that the GovReady-Q always runs the template on the most recent, current set of answers to the Module's questions. As a result, there is no database table for output documents.

6.2.4 Database Query Examples

Example: Find all approved answers to a particular question across users and tasks

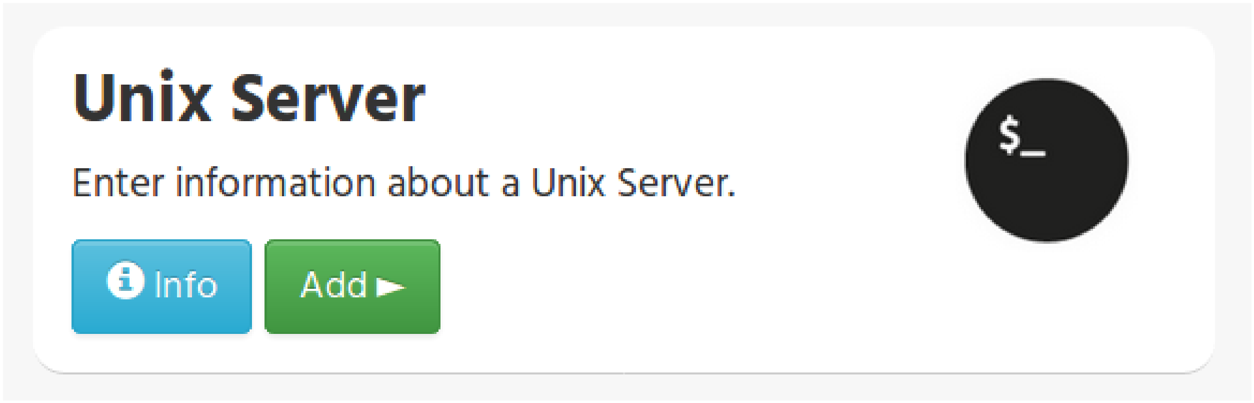
Scenario: Unix File Server App contains a text-type question named "Hostname". Many users have finished answering all of the questions in the app. However, our reviewers have only approved some of the answers so far. I want to write an SQL query to return all approved answers to the "Hostname" question.

In this section, we will build up an SQL query to extract the data identified in the scenario. The query will be built progressively over the next several sections to explain the rationale behind the GovReady-Q data model. Some of GovReady-Q's design choices — including separating the definitions of compliance apps from user-submitted data, as well as recording an immutable history of user answers — are reflected in the SQL queries below. The complete SQL query is shown at the end.

You may prefer to use the GovReady-Q API instead of writing a low-level database query, but this example is illustrative for understanding GovReady-Q's data model no matter which method you use to query the data.

Find the AppVersions

First locate the `AppSource` "slug" and `AppVersion` "appname" that identifies a compliance app in GovReady-Q's database. Find the app in the compliance apps catalog and click its *Info* button:



The slug and the appname of the compliance app can be found in the URL:

> http://mygovreadyq/store/**myapps/unix_file_server**

In this case the slug is "myapps" and the appname is "unix_file_server". These two fields identify the compliance app across its versions.

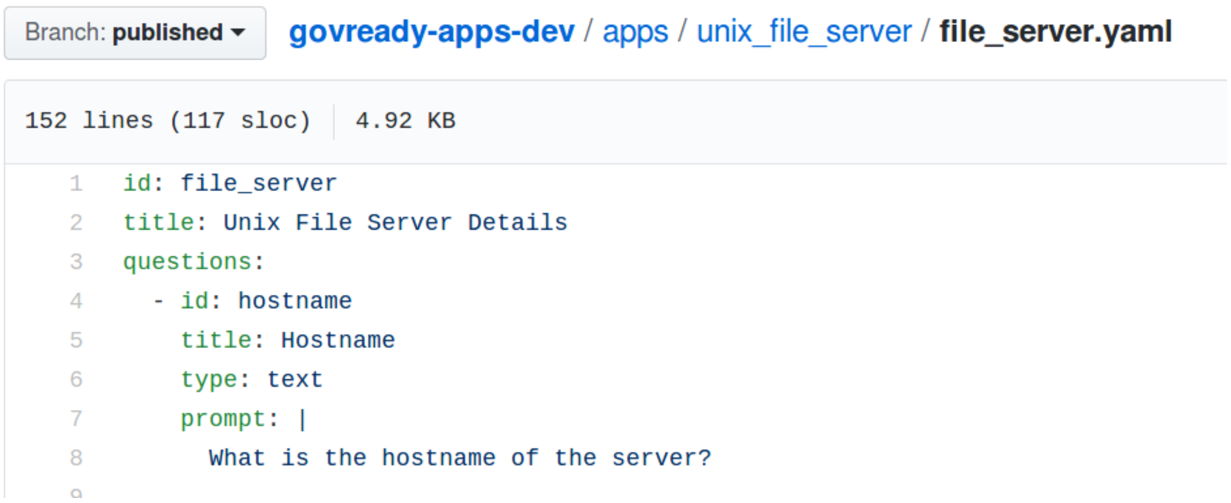
Construct an SQL query to return the numeric IDs of the AppVersions in the database for this compliance app. Each AppVersion may be a different version of the compliance app or a different instance of the same app in use by different users.

```
SELECT guidedmodules_appversion.id
FROM guidedmodules_appversion
LEFT JOIN guidedmodules_appsource
       ON guidedmodules_appsource.id = guidedmodules_appversion.source_id
WHERE guidedmodules_appsource.slug = "myapps"
  AND guidedmodules_appversion.appname = "unix_file_server";
```

This query will be adapted in the next section to find the hostname question.

Find the ModuleQuestions

Consult the compliance app source code YAML files to determine the "module_name" of the `Module` and "key" of the `ModuleQuestion` — which are in the "id" fields in the YAML file.


Branch: published
govready-apps-dev / apps / unix_file_server / file_server.yaml
152 lines (117 sloc) | 4.92 KB

```yaml
id: file_server
title: Unix File Server Details
questions:
  - id: hostname
    title: Hostname
    type: text
    prompt: |
      What is the hostname of the server?
```

The `Module` containing the hostname question has "file_server" as its module_name, and the `ModuleQuestion`'s key is "hostname."

Construct a preliminary SQL query to find all of the `ModuleQuestion` records for this question:

```sql
SELECT guidedmodules_modulequestion.id
FROM guidedmodules_modulequestion
LEFT JOIN guidedmodules_module
       ON guidedmodules_module.id = guidedmodules_modulequestion.module_id
WHERE guidedmodules_module.module_name = "file_server"
  AND guidedmodules_modulequestion.key = "hostname";
```

This query might be too broad — it does not restrict the questions to those defined in the Unix File Server compliance app. There might be other compliance apps that use the same module_name and question key. Combine the first two queries to ensure only questions in the Unix File Server app are returned using a LEFT JOIN to bridge the tables:

```sql
SELECT guidedmodules_modulequestion.id
FROM guidedmodules_modulequestion
LEFT JOIN guidedmodules_appsource
       ON guidedmodules_appsource.id = guidedmodules_appversion.source_id
LEFT JOIN guidedmodules_appversion
       ON guidedmodules_appversion.id = guidedmodules_module.app_id
LEFT JOIN guidedmodules_module
       ON guidedmodules_module.id = guidedmodules_modulequestion.module_id
WHERE guidedmodules_appsource.slug = "myapps"
  AND guidedmodules_appversion.appname = "unix_file_server"
  AND guidedmodules_module.module_name = "file_server"
  AND guidedmodules_modulequestion.key = "hostname";
```

We'll call this query `MODULE_QUESTIONS` — we'll use it as a sub-query in the next step.

Find the history of answers

GovReady-Q has been designed so that separate tables contain the definition of the question and the user-submitted answers to the question. Each answer is connected to a `ModuleQuestion` through a `TaskAnswer`. Locate the TaskAnswers for the questions:

```
SELECT guidedmodules_taskanswer.id
FROM guidedmodules_taskanswer
WHERE guidedmodules_taskanswer.question_id IN (MODULE_QUESTIONS);
```

Replace `MODULE_QUESTIONS` with the preceding SQL query, inserting it as a sub-query.

The TaskAnswer table does not hold user answers, however. Answers are stored in the `TaskAnswerHistory` table where the complete history of answers to questions are stored. We'll now adapt the query to fetch the history of answers to this question, including some metadata about the answers, by using a LEFT JOIN to bridge the TaskAnswerHistory table and the TaskAnswer table:

```
SELECT guidedmodules_taskanswer.id, answer.stored_value, answer.created, siteapp_user.
↪username
FROM guidedmodules_taskanswerhistory AS answer
LEFT JOIN guidedmodules_taskanswer
       ON guidedmodules_taskanswer.id = answer.taskanswer_id
LEFT JOIN siteapp_user
       ON siteapp_user.id = answer.answered_by_id
WHERE guidedmodules_taskanswer.question_id IN (MODULE_QUESTIONS);
```

Here is an example result:

Task Answer	Stored Value	Created	Username
10	"server1.company.com"	2018-05-19 20:33	user1
10	"server2.company.com"	2018-05-20 10:15	user1
10	null	2018-05-20 10:35	user1
11	"server2.company.com"	2018-05-19 16:20	user2

This is the complete history of answers for the "hostname" question in two separate Tasks, i.e. two instantiations of the compliance app started by different users. The two instantiations of the question are identified by their `TaskAnswer` "id"s, 10 and 11.

The history for TaskAnswer 10 has three rows. Two rows – the first two — reflect old answers to questions. This indicates the user returned to the question twice. On the first occasion, the user replaced the original answer with `"server2.company.com"`. On the second revisit, the user replaced the original answer with `null`, clearing the answer because the user decided they didn't know the answer or the question didn't apply to them.

The second TaskAnswer was answered once.

We'll adapt this query in the next step to fetch just the current (most recent) answer in each Task.

Find the current answer to each question

The current answer for each question is stored in the `TaskAnswerHistory` record with the highest "id" for each TaskAnswer. The IDs in the TaskAnswerHistory table are assigned strictly sequentially. To determine which TaskAnswerHistory record holds the current answer, use `GROUP BY` and `max` to fetch one TaskAnswerHistory for each TaskAnswer:

```
SELECT max(answer.id)
FROM guidedmodules_taskanswerhistory AS answer
LEFT JOIN guidedmodules_taskanswer
       ON guidedmodules_taskanswer.id = answer.taskanswer_id
LEFT JOIN siteapp_user
       ON siteapp_user.id = answer.answered_by_id
```

(continues on next page)

(continued from previous page)

```
WHERE guidedmodules_taskanswer.question_id IN (MODULE_QUESTIONS)
GROUP BY guidedmodules_taskanswer.id;
```

id
103
104

This result holds the current answers to the hostname question. We'll call this query `CURRENT_ANSWERS` — we'll use it as a sub-query in the next query.

To fetch the answers and metadata but for the current answers, we'll query the `TaskAnswerHistory` table using the `CURRENT_ANSWERS` query as a sub-query to identify just the rows that are current answers to questions:

```
SELECT taskanswer_id, stored_value, created, username, reviewed
FROM guidedmodules_taskanswerhistory
LEFT JOIN siteapp_user
       ON siteapp_user.id = answered_by_id
WHERE guidedmodules_taskanswerhistory.id IN (CURRENT_ANSWERS);
```

Here is an example result:

Task Answer	Stored Value	Created	Username	Reviewed
10	null	2018-05-20 10:35	user1	0
11	"server2.company.com"	2018-05-19 16:20	user2	2

This result holds the current answers to the Unix File Server hostname question across all instances of the compliance app in the GovReady-Q installation. Notice that the rows in the previous table that represented replaced answers to the first TaskAnswer are omitted from the results in this query and only the current answer for each `Task` is included.

The "stored_value" column holds the user's answer encoded in JSON. In JSON, text (strings) are enclosed in double quotes. Therefore we know that the second answer is text. In JSON, `null` (without double quotes around it) represents an empty value — in GovReady-Q, that means the user skipped the question choosing *I Don't Know*, *It Doesn't Apply*, or *I'll Come Back*.

We'll modify this query in the next section to filter on the reviewed status of each answer.

Filter on approved answers

The "reviewed" field of `TaskAnswerHistory` stores GovReady-Q's simple workflow status of the answer. The values are `0` (not reviewed), `1` (reviewed), and `2` (approved). To select just approved answers, add a WHERE clause to the previous SQL query:

```
SELECT taskanswer_id, stored_value, created, username, reviewed
FROM guidedmodules_taskanswerhistory
LEFT JOIN siteapp_user
       ON siteapp_user.id = answered_by_id
WHERE guidedmodules_taskanswerhistory.id IN (CURRENT_ANSWERS)
  AND reviewed = 2;
```

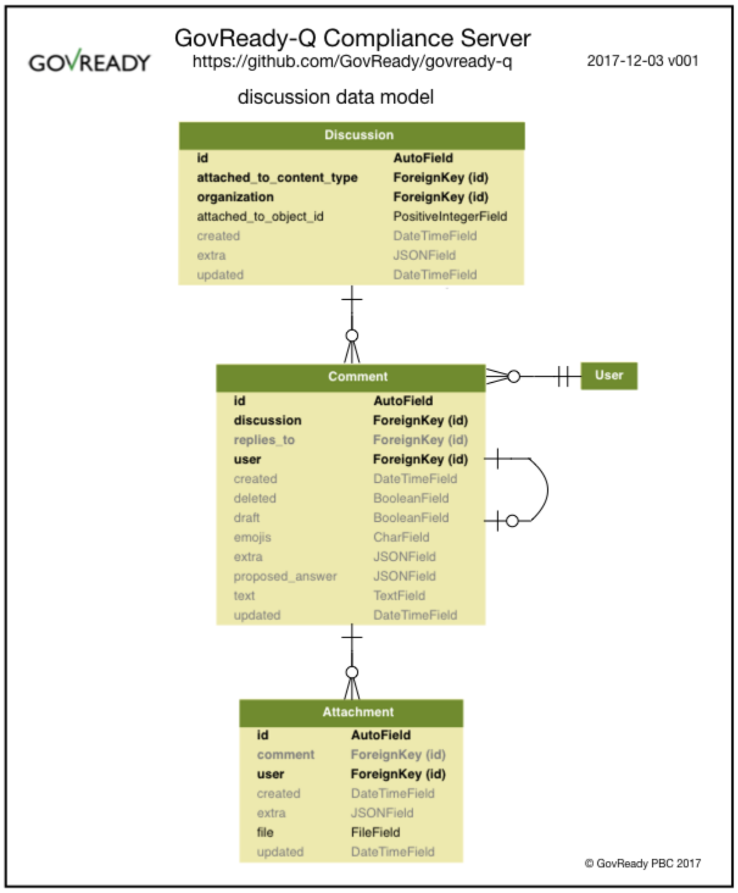
The query extracts the answers in a structure similar to the following table:

Task Answer	Stored Value	Created	Username	Reviewed
11	"server2.company.com"	2018-05-19 16:20	user2	2

This is the complete query to extract the approved answers to the hostname question in the Unix File Server compliance app. The query has been simplified by replacing a sub-query with `CURRENT_ANSWERS`, which itself has a sub-query that has been replaced by `MODULE_QUESTIONS`. Both sub-queries can be found above.

6.3 Discussions

The diagram below provides a summary representation of GovReady-Q's Django `discussion` data model that handles discussions, comments, and invitations.



A single discussion can be instantiated and associated to any task (task ~= "question"). A discussion can have multiple comments. Comments can have multiple attachments.

6.4 Generating Detailed Data Models

Below are instructions to use `django-extensions` to generate detailed data models.

```
# Install django-extensions
# http://django-extensions.readthedocs.io/en/latest/installation_instructions.html
apt install graphviz-dev
pip3 install django-extensions pygraphviz
```

(continues on next page)

(continued from previous page)

```
# Add django-extensions INSTALLED_APPS in siteapp > settings.py
# INSTALLED_APPS = (
#     ...
#     'django_extensions',
# )

# examples:
python3 manage.py graph_models -a -g -o my_project_visualized.png
python3 manage.py graph_models -a -o my_project.png
python3 manage.py graph_models -a > my_project.dot
# for a single django app:
python3 manage.py graph_models app1 -o my_project_app1.png
```

CHAPTER

SEVEN

TESTING

7.1 Running Tests

GovReady-Q's unit tests and integration tests are currently combined. Our integration tests uses Selenium to simulate user interactions with the interface.

To run the integration tests, you'll also need to install chromedriver:

```
sudo apt-get install chromium-chromedriver   (on Ubuntu)
brew cask install chromedriver               (on Mac)
```

Navigate within your terminal to GovReady-Q top level directory.

Then run the test suite with:

```
./manage.py test
```

NOTE: Depending on your Python3 configuration, you may need to run:

```
python3 manage.py test
```

To selectively run tests from individual modules:

```
# test rendering of guided modules
./manage.py test guidedmodules

# test general siteapp logic
./manage.py test siteapp

# test discussion functionality
./manage.py test discussion
```

Or to selectively run tests from individual classes or methods:

```
# run tests from individual test class
./manage.py test siteapp.tests.GeneralTests

# run tests from individual test method
./manage.py test siteapp.tests.GeneralTests.test_login
```

7.2 Test Coverage Report

To produce a code coverage report, run the tests with `coverage`:

```
coverage run --source='.' --branch manage.py test
coverage report
```

7.3 Code Scanning and Analysis

GovReady-Q is a Python web application written on top of the Django framework and uses a variety of industry standard Javascript libraries. See Software Requirements for high level view and the `requirement*.txt` files for detailed view.

GovReady-Q's Python application code is found in the `*.py` files in the following directories and their subdirectories:

- discussion/
- guidedmodules/
- siteapp/

The small `manage.py` script in the root directory is part of the Django framework. We use bash utilities scripts (`*.sh`) to automate installation and maintenance tasks of the code base. Python scripts in `.circleci` directory are used within our Continuous Implementation pipeline.

7.3.1 Simple Static Code Analysis

To run a static code analysis with our typical settings:

```
bandit -s B101,B110,B603 -r discussion/ guidedmodules/ siteapp/
```

We use `-s` on the command-line and `nosec` in limited places in the source code to disable some checks that are determined after review to be false positives.

7.3.2 Detailed Static and Dynamic Code Analysis

We periodically scan GovReady-Q's code base with more traditional/powerful tools and remediate critical and high vulnerabilities.

To scan GovReady-Q's codebase, you will need to configure your tools to scan Python code. You are looking for the `*.py` files across the code base.

To scan or do other penetration tests on the code base, we recommend deploying GovReady-Q with Docker.

7.4 Dependency Management and Vulnerability Testing

Our `requirements.txt` file is designed to work with `pip install --require-hashes`, which ensures that every installed dependency matches a hash stored in this repository. The option requires that every dependency (including dependencies of dependencies) be listed, pinned to a version number, and paired with a hash. We therefore don't manually edit `requirements.txt`. Instead, we place our immediate dependencies in `requirements.in` and run `requirements_txt_updater.sh` (which calls pip-tools's pip-compile command) to update the `requirements.txt` file for production.

Continuous integration is set up with CircleCI at https://circleci.com/gh/GovReady/govready-q and performs unit tests, integration tests, and security checks on our dependencies.

1. CI runs `requirements_txt_checker.sh` which ensures `requirements.txt` is in sync with `requirements.in`. This script is set up to run against any similar files as well, such as MySQL-specific `requirements_mysql.*` files.
2. CI checks that there are no known vulnerabilities in the dependencies using pyup.io.
3. CI checks that all packages are up to date with upstream sources (unless the package and its latest upstream version are listed in `requirements_txt_checker_ignoreupdates.txt`).

7.5 Populating sample data for manual testing and verification

In some cases, you may wish to perform manual testing on an instance of GovReady-Q which has been populated with data. Several Django commands have been added to facilitate this, in the `testmocking` module. Generated data is intended to be structurally similar to what might be found in a real GovReady-Q instance, but the actual content of the data will often appear machine-generated.

If you wish to get up and running quickly, the following command is recommended:

```
python3 manage.py add_data --non-interactive
```

The `add_data` command will fill in a recommended set of sample data.

CHAPTER EIGHT

VERSION 0.9.0

8.1 What's New in 0.9.0

Release 0.9.0 (coming Autumn 2019) is a minor release improving the user experience and performance.

- Faster loading and launching of Assessments/questionnaires
- Simplified install with no subdomains to worry about
- Replaces subdomain multi-tenancy with simplified "Portfolios" model
- Improved authoring screens
- Helpful new start page
- Adds and updates portfolio level permissions

Release 0.9.0 removes multi-tenancy and serves all pages from the same domain. In the earlier multi-tenancy versions, requests to GovReady-Q came in on subdomains: the subdomain determined which organization to associate with the request, and individuals had to re-login across subdomains. Little value was being realized by actual users for the subdomain-based multi-tenancy. Removing subdomain-based multi-tenancy reduces technical debt simplifies deployment.

This release's compliance apps catalog now reads from the database rather than constantly rescanning remote repositories and the file system. The app catalog cache is removed since the page loads much faster. Release 0.9.0 begins to replace the "compliance app" terminology with the plain language "projects" and "assessment" terminology in end user pages.

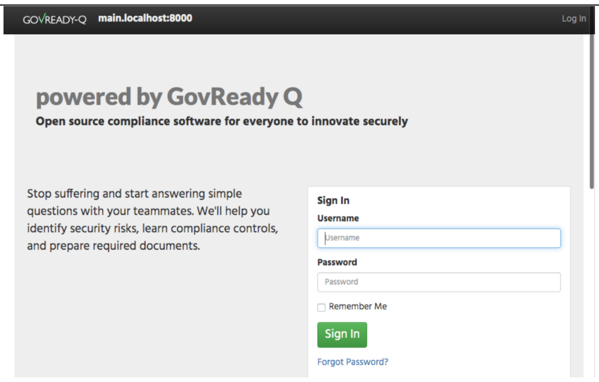
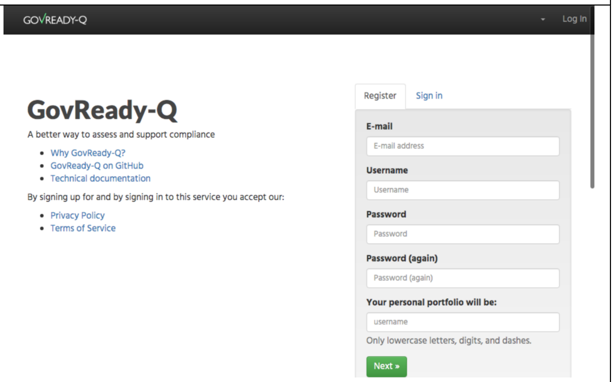
This release also introduces a "Portfolios" feature to organize and manage related projects.

For a complete list of changes see the 0.9.0.dev branch CHANGELOG.

Release 0.9.0 progress can be found on the `0.9.0.dev` and `0.9.0.rc-xxx` branches.

8.2 Screenshots

8.2.1 Sign-in Page

0.8.6 (Old Version)	0.9.0 (New Version)
	

GOVREADY-Q main.localhost:8000 Log in

powered by GovReady Q

Open source compliance software for everyone to innovate securely

Stop suffering and start answering simple questions with your teammates. We'll help you identify security risks, learn compliance controls, and prepare required documents.

Sign In

Username

Password

Remember Me

Sign In

Forgot Password?

GOVREADY-Q Log in

GovReady-Q

A better way to assess and support compliance

- Why GovReady-Q?
- GovReady-Q on GitHub
- Technical documentation

By signing up for and by signing in to this service you accept our:

- Privacy Policy
- Terms of Service

Register Sign in

E-mail

Username

Password

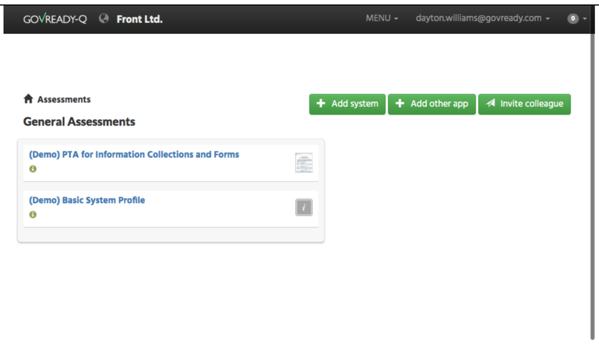
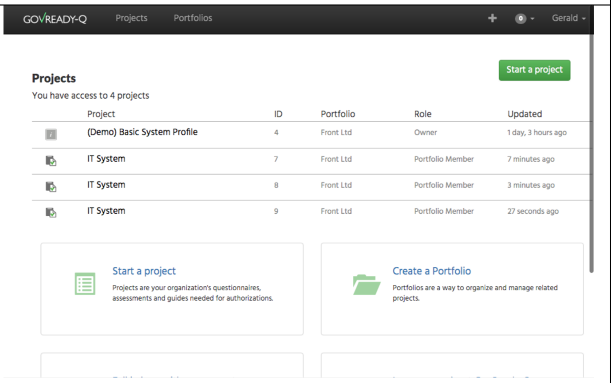
Password (again)

Your personal portfolio will be:

Only lowercase letters, digits, and dashes.

Next »

8.2.2 Projects List Page

0.8.6 (Old Version)	0.9.0 (New Version)
	

GOVREADY-Q Front Ltd. MENU dayton.williams@govready.com

Assessments

General Assessments

Add system Add other app Invite colleague

(Demo) PTA for Information Collections and Forms

(Demo) Basic System Profile

GOVREADY-Q Projects Portfolios Gerald

Projects

You have access to 4 projects

Start a project

	Project	ID	Portfolio	Role	Updated
	(Demo) Basic System Profile	4	Front Ltd	Owner	1 day, 3 hours ago
	IT System	7	Front Ltd	Portfolio Member	7 minutes ago
	IT System	8	Front Ltd	Portfolio Member	3 minutes ago
	IT System	9	Front Ltd	Portfolio Member	27 seconds ago

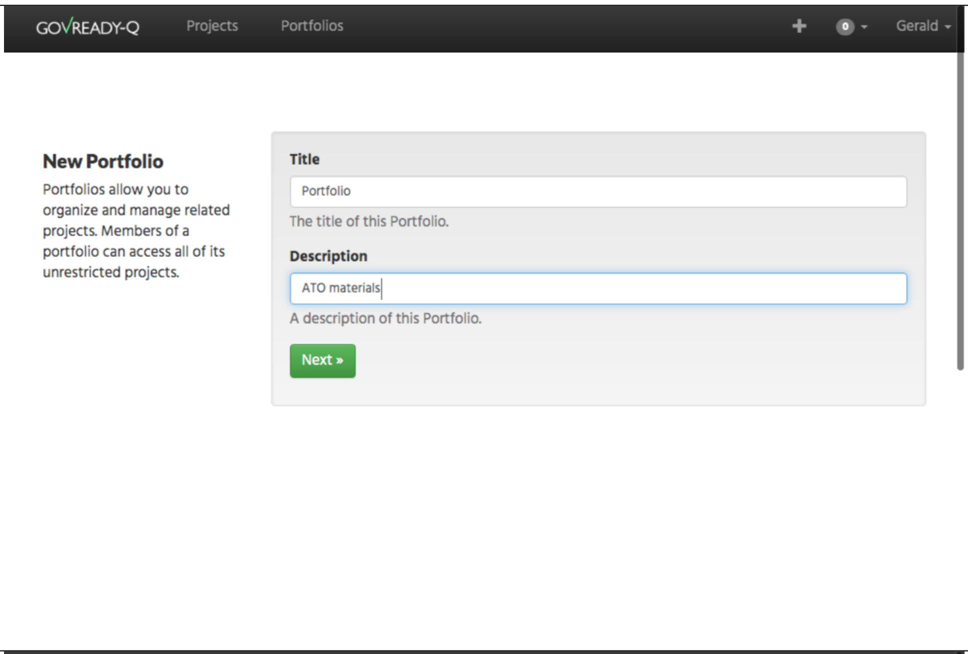
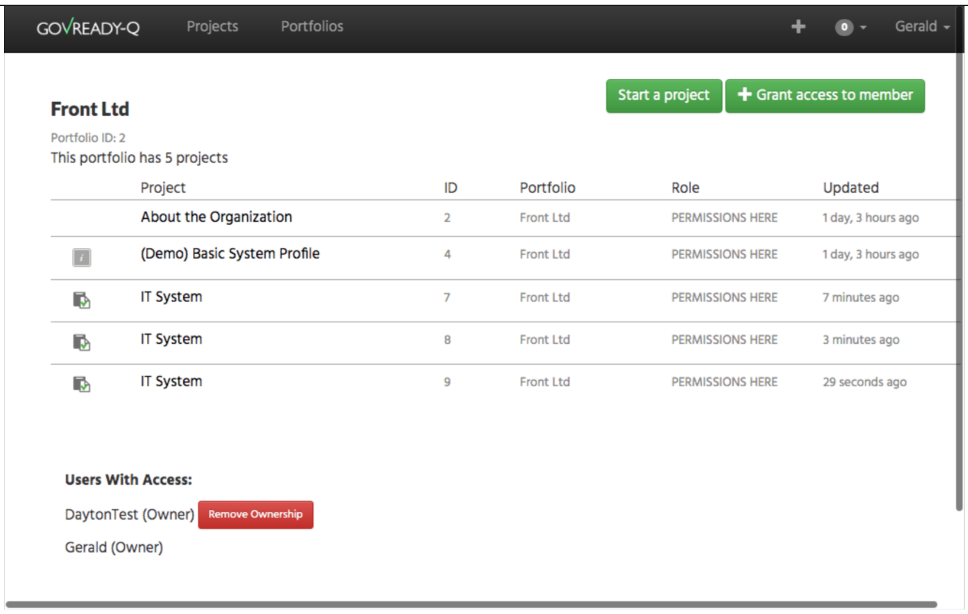
Start a project

Projects are your organization's questionnaires, assessments and guides needed for authorizations.

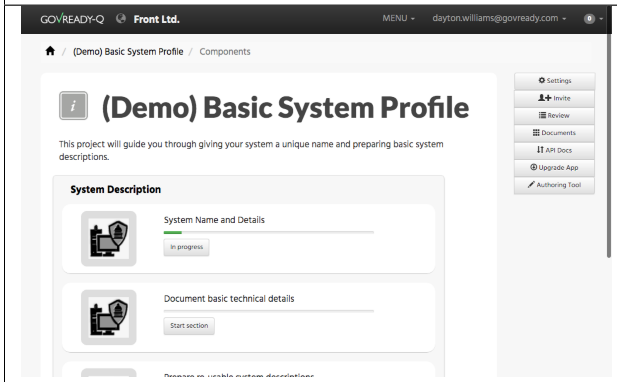
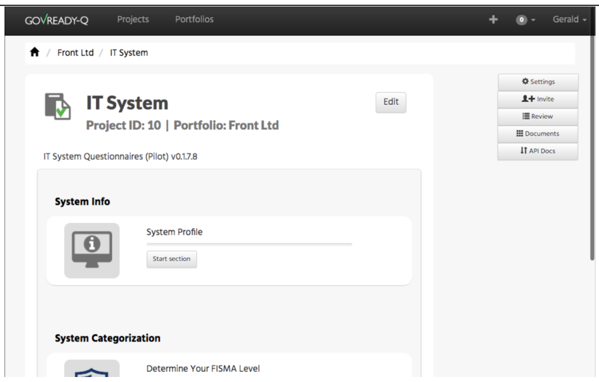
Create a Portfolio

Portfolios are a way to organize and manage related projects.

8.2.3 Portfolios (New in 0.9.0)

0.8.6 (Old Version)	0.9.0 (New Version)
Feature did not exist	 GOVREADY-Q Projects Portfolios Gerald **New Portfolio** Portfolios allow you to organize and manage related projects. Members of a portfolio can access all of its unrestricted projects. **Title** Portfolio The title of this Portfolio. **Description** ATO materials A description of this Portfolio. Next »
Feature did not exist	 GOVREADY-Q Projects Portfolios Gerald Start a project Grant access to member **Front Ltd** Portfolio ID: 2 This portfolio has 5 projects Project | ID | Portfolio | Role | Updated About the Organization | 2 | Front Ltd | PERMISSIONS HERE | 1 day, 3 hours ago (Demo) Basic System Profile | 4 | Front Ltd | PERMISSIONS HERE | 1 day, 3 hours ago IT System | 7 | Front Ltd | PERMISSIONS HERE | 7 minutes ago IT System | 8 | Front Ltd | PERMISSIONS HERE | 3 minutes ago IT System | 9 | Front Ltd | PERMISSIONS HERE | 29 seconds ago **Users With Access:** DaytonTest (Owner) Remove Ownership Gerald (Owner)

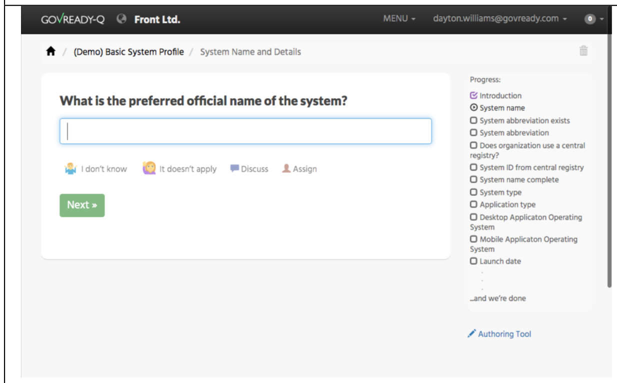
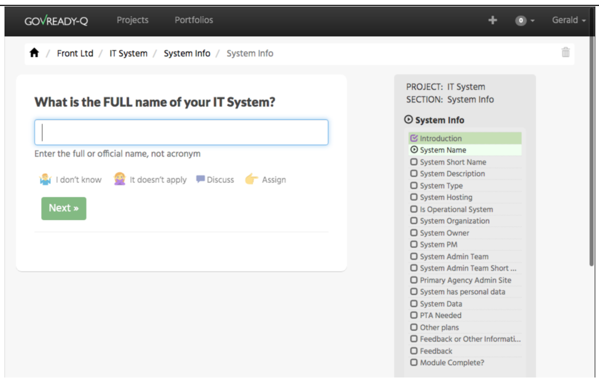
8.2.4 Module View

0.8.6 (Old Version)	0.9.0 (New Version)
	

GOVREADY-Q
Front Ltd.
MENU
dayton.williams@govready.com
(Demo) Basic System Profile / Components
(Demo) Basic System Profile
This project will guide you through giving your system a unique name and preparing basic system descriptions.
System Description
System Name and Details
In progress
Document basic technical details
Start section
Settings
Invite
Review
Documents
API Docs
Upgrade App
Authoring Tool

GOVREADY-Q
Projects
Portfolios
Gerald
Front Ltd / IT System
IT System
Project ID: 10 | Portfolio: Front Ltd
Edit
IT System Questionnaires (Pilot) v0.1.7.8
System Info
System Profile
Start section
System Categorization
Determine Your FISMA Level
Settings
Invite
Review
Documents
API Docs

8.2.5 Question Page

0.8.6 (Old Version)	0.9.0 (New Version)
	

GOVREADY-Q
Front Ltd.
MENU
dayton.williams@govready.com
(Demo) Basic System Profile / System Name and Details
What is the preferred official name of the system?
I don't know
It doesn't apply
Discuss
Assign
Next »
Progress:
Introduction
System name
System abbreviation exists
System abbreviation
Does organization use a central registry?
System ID from central registry
System name complete
System type
Application type
Desktop Applicaton Operating System
Mobile Applicaton Operating System
Launch date
...and we're done
Authoring Tool

8.3 Release Date

The target release date for 0.9.0 is Autumn 2019.

8.4 Upgrading to 0.9.0 from 0.8.x

Backup your database before upgrading to 0.9.0. Release 0.9.0 performs database changes that makes rolling back difficult.

See Migration Guide for GovReady-Q (0.8.6 to 0.9.0).

8.5 Installing 0.9.0

Release 0.9.0 simplifies the installation of GovReady-Q by removing the need to manage subdomain-based multi-tenancy.

Three example assessments are preloaded during the install of 0.9.0 to demonstrate the experience of using GovReady-Q.

The installation of assessments (aka "Compliance Apps") requires an extra step in 0.9.0 but greatly improves user experience.

Click one of the tab belows to see quickstart for indicated platform.

Docker

Installing with Docker

Make sure you first install Docker (https://docs.docker.com/engine/installation/) and, if appropriate, grant non-root users access to run Docker containers (https://docs.docker.com/engine/installation/linux/linux-postinstall/#manage-docker-as-a-non-root-user) (or else use *sudo* when invoking Docker below).

Start

```
# Create a local directory for authoring Q assessment files
mkdir -p /path/to/govready-q-files
# mkdir -p /codedata/code/govready-q-files

# Run the docker container in detached mode
docker container run --detach --name govready-q -p 127.0.0.1:8000:8000 \
-e HOST=localhost -e PORT=8000 -e HTTPS=false -e DBURL= -e DEBUG=true \
govready/govready-q-0.9.0.dev

# Create admin account and organization data if setting up a new database
docker container exec -it govready-q first_run

# Stop, start container (when needed)
docker container stop govready-q
docker container start govready-q

# View logs - useful if site does not appear
docker container logs govready-q

# To destroy the container and all user data entered into Q
docker container rm -f govready-q
```

Visit your GovReady-Q site in your web browser at:

> http://localhost:8000/

Dockerized GovReady-Q supports additional options. Below is a more complete example

parameters available.

```
# Run the docker container in detached mode
docker container run --detach --name govready-q -p 127.0.0.1:8000:8000 \
-e HOST=localhost -e PORT=8000 -e HTTPS=false -e DBURL= -e DEBUG=true \
-e EMAIL_HOST= -e EMAIL_PORT= -e EMAIL_USER= -e EMAIL_PW= -e EMAIL_DOMAIN= \
```

(continues on next page)

(continued from previous page)

```
--mount type=bind,source="$(pwd)",target=/mnt/q-files-host \
govready/govready-q-0.9.0.dev
```

Alternatively, we offer a shell script that helps launch GovReady-Q. Do the following to download the shell script and launch GovReady-Q docker container.

```
cd /path/to/working/dir

# Get the docker_container_run.sh shell script
wget https://raw.githubusercontent.com/GovReady/govready-q/master/deployment/docker/
↪docker_container_run.sh

# Make it executable
chmod +x docker_container_run.sh

./docker_container_run.sh --name govready-q --relaunch --debug -v --appsdevdir /
↪codedata/code/govready-q-files --image govready/govready-q-0.9.0.dev
```

Logs for Debugging

The container's console shows the output of container's start-up commands including database migrations and process startup. The container's console log can be accessed with

```
docker container logs govready-q-0.9.0
```

GovReady-Q application logs can be found in */var/log* within the container to track status and assist with debugging. These files contain access logs and other program output.

- */var/log/application-stderr.log* - GovReady-Q application standard error
- */var/log/application-stdout.log* - GovReady-Q application standard out
- */var/log/notificationemails-stderr.log* - GovReady-Q email notifications standard error
- */var/log/notificationemails-stdout.log* - GovReady-Q email notifications standard error
- */var/log/supervisord.log* - Supervisor daemon

Debugging "Internal Server Error" Messages

A special management command can be used to see the application log files to debug unhandled "Internal Server Error" (HTTP code 500) messages displayed in the browser to end users

```
docker container exec govready-q-0.9.0 tail_logs
# Replace "govready-q-0.9.0" with name of your container or use container id
```

tail_logs takes the same arguments as Unix *tail*. For instance, add *-n 1000* to see the most recent 1,000 log lines, or add *-f* to continue to output the logs as the log files grow.

```
# most recent 200 lines of logs
docker container exec govready-q-0.9.0 tail_logs -n 200

# real-time display of logs
docker container exec govready-q-0.9.0 tail_logs -f
```

The log files can also be accessed by mounting */var/log* with a Docker bind-mount or as a volume (and that's the only way to see the logs if *docker container exec* cannot be used in your environment).

macOS

Installing on macOS

GovReady-Q calls requires Python 3.6 or higher to run and several Unix packages to provide full functionality. Install the Homebrew package manager (https://brew.sh) to easily install Unix packages on macOS. Homebrew will install all packages in your userspace and not change native macOS Python or other libraries.

```
# install Homebrew package manager
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Now install Python3 and the required Unix packages.

```
# install dependencies using brew
brew install python3

# install other packages:
brew install unzip graphviz pandoc selenium-server-standalone magic libmagic
brew cask install wkhtmltopdf
```

Installing GovReady-Q

Clone GovReady-Q source code and install.

```
# clone GovReady-Q
git clone https://github.com/govready/govready-q
cd govready-q

# checkout the 0.9.0.dev (or desired 0.9.0.rc-0xx branch)
git checkout 0.9.0.dev

# install Python 3 packages
pip3 install --user -r requirements.txt

# install Bootstrap and other vendor resources locally
./fetch-vendor-resources.sh
```

Run the final setup commands to initialize a local Sqlite3 database in *local/db.sqlite* to make sure everything is OK so far:

```
# run database migrations (sqlite lite database used by default)
python3 manage.py migrate

# load a few critical modules
python3 manage.py load_modules

# create superuser with initial account if setting up a new database
# skip if you pass environment variables to connect to a persistent database
python3 manage.py first_run
```

Start GovReady-Q

```
# run the server
python3 manage.py runserver
```

Visit your GovReady-Q site in your web browser at:

http://localhost:8000/

RHEL/CentOS 7

Installing on RHEL/CentOS 7

Instructions applicable RHEL 7, CentOS 7 and Amazon Linux 2.

GovReady-Q calls requires Python 3.6 or higher to run and several Linux packages to provide full functionality.

```
# if necessary, enable EPEL and IUS repositories
rpm -i https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm https://
↪rhel7.iuscommunity.org/ius-release.rpm

# install dependencies
sudo yum install \
unzip python36-pip python36-devel \
graphviz \
pandoc xorg-x11-server-Xvfb wkhtmltopdf \

# optional install gcc to build the uWSGI Python package.
sudo yum install gcc

# optional insall of postgress and/or mysql
sudo yum install postgresql mysql-devel
```

GovReady-Q calls out to *git* to fetch apps from git repositories, but that requires git version 2 or later because of the use of the GIT_SSH_COMMAND environment variable. RHEL stock git is version 1. Switch it to version 2+ by using the IUS package:

```
# if necessary, remove any git currently installed
yum remove git
# install git2u
yum install git2u
```

Installing GovReady-Q

Clone GovReady-Q source code and install.

```
# clone GovReady-Q
git clone https://github.com/govready/govready-q
cd govready-q

# checkout the 0.9.0.dev (or desired 0.9.0.rc-0xx branch)
git checkout 0.9.0.dev

# install Python 3 packages
pip3 install --user -r requirements.txt

```

(continues on next page)

(continued from previous page)

```
# install Bootstrap and other vendor resources locally
./fetch-vendor-resources.sh
```

Run the final setup commands to initialize a local Sqlite3 database in *local/db.sqlite* to make sure everything is OK so far:

```
# run database migrations (sqlite lite database used by default)
python3 manage.py migrate

# load a few critical modules
python3 manage.py load_modules

# create superuser with initial account
# skip if you pass environment variables to connect to a persistent database
python3 manage.py first_run

python3 manage.py first_run
```

Start GovReady-Q

```
# run the server
python3 manage.py runserver
```

Visit your GovReady-Q site in your web browser at:

http://localhost:8000/

Ubuntu 16.04

Installing on Ubuntu

Instructions provide basic guidance on setting up GovReady-Q on an Ubuntu 16.04 server with Nginx. These commands should be run from the root directory of the GovReady-Q code repository.

GovReady-Q calls requires Python 3.6 or higher to run and several Linux packages to provide full functionality.

```
# upgrade apt-get
apt-get update && apt-get upgrade -y

# install dependencies
apt-get install -y \
  unzip \
  python3 python-virtualenvpython3-pip \
  python3-yaml \
  nginx uwsgi-plugin-python3supervisor \
  memcached \
  graphviz

# optional install gcc to build the uWSGI Python package.
sudo yum install gcc

# optional insall of postgress and/or mysql
apt-get install -y postgresql mysql-devel
```

Installing GovReady-Q

Clone GovReady-Q source code and install.

```
# clone GovReady-Q
git clone https://github.com/govready/govready-q
cd govready-q

# checkout the 0.9.0.dev (or desired 0.9.0.rc-0xx branch)
git checkout 0.9.0.dev

# install Python 3 packages
pip3 install --user -r requirements.txt

# install Bootstrap and other vendor resources locally
./fetch-vendor-resources.sh
```

Run the final setup commands to initialize a local Sqlite3 database in *local/db.sqlite* to make sure everything is OK so far:

```
# run database migrations (sqlite lite database used by default)
python3 manage.py migrate

# load a few critical modules
python3 manage.py load_modules

# create superuser with initial account
# skip if you pass environment variables to connect to a persistent database
python3 manage.py first_run
```

Start GovReady-Q

```
# run the server
python3 manage.py runserver
```

Visit your GovReady-Q site in your web browser at:

http://localhost:8000/

Windows

Installing on Windows (with Docker)

GovReady-Q can only be installed on Windows using Docker.

Make sure you first install Docker (https://docs.docker.com/docker-for-windows/install/).

Start

```
# Run the docker container in detached mode
docker container run --name govready-q --detach -p 8000:8000 govready/govready-q-0.9.
↪0.dev

# Create admin account and organization data
```

(continues on next page)

(continued from previous page)

```
# Skip if you pass environment variables to connect to a persistent database
docker container exec -it govready-q first_run

# Stop, start container
docker container stop govready-q
docker container start govready-q

# View logs - useful if site does not appear
docker container logs govready-q

# To destroy the container and all user data entered into Q
docker container rm -f govready-q
```

Visit your GovReady-Q site in your web browser at:

> http://localhost:8000/

8.6 Upgrading to 0.9.0 from 0.8.x

Backup your database before upgrading to 0.9.0. Release 0.9.0 performs database changes that makes rolling back difficult.

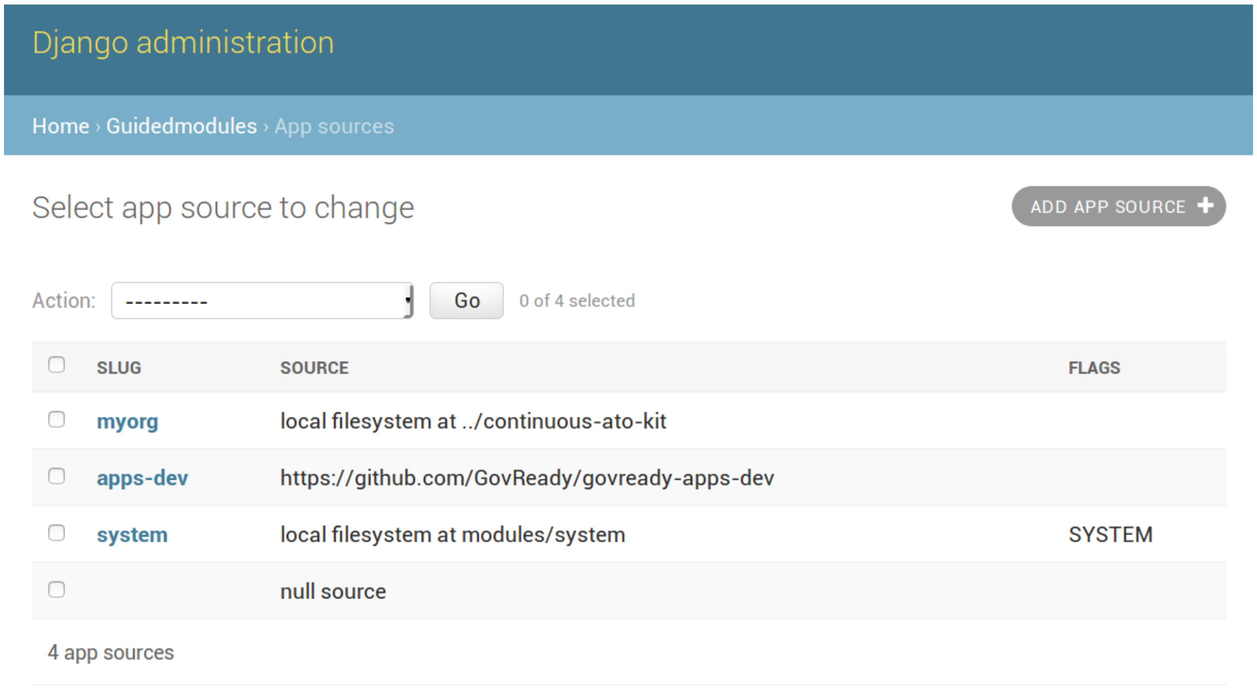
See Migration Guide for GovReady-Q (0.8.6 to 0.9.0).

8.7 Adding and Managing "Compliance Apps" in 0.9.0

GovReady-Q wll load a small set of example questionnaires as part of the `python manage.py first_run` command during installation.

GovReady-Q must be configured by an administrator to load additional compliance apps (e.g., assessments and questionnaires) from one or more sources, which can be local directories or remote git repositories. Full administrative privileges are assigned to original user account created when executing `python manage.py first_run` during installation.

App Sources are configured in the Django admin at the URL `/admin` on your GovReady-Q domain under `App Sources`:



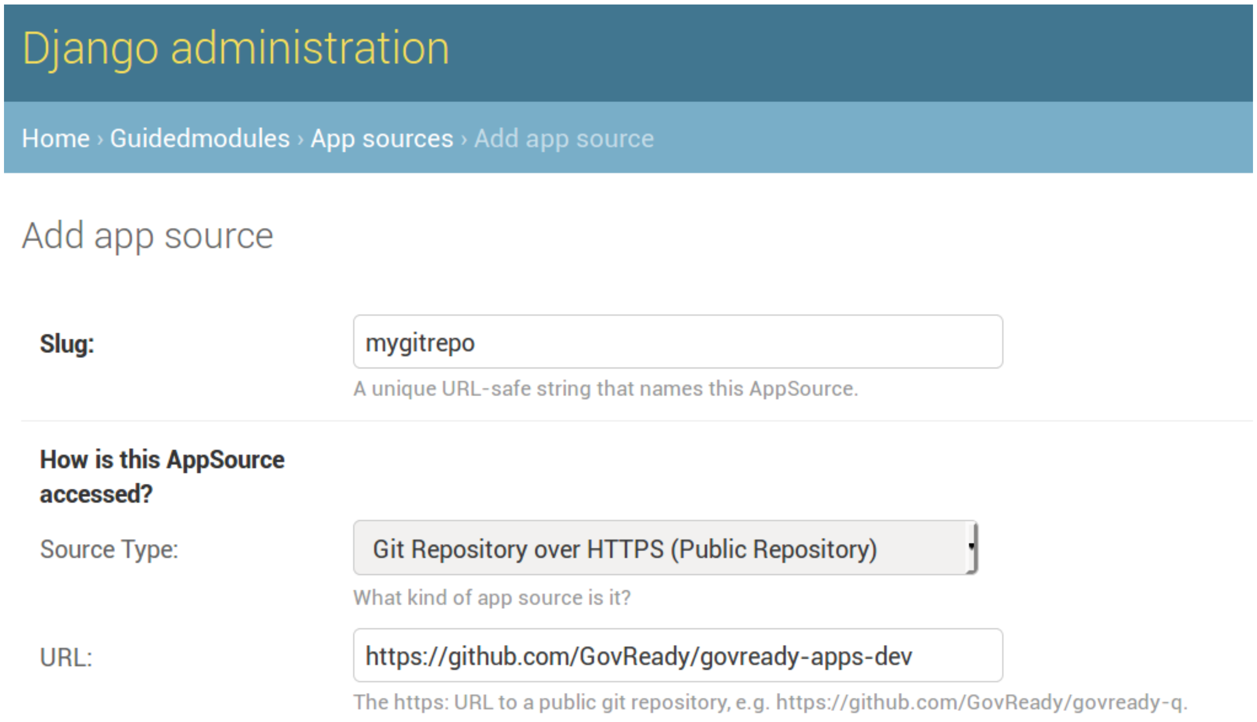
Select app source to change

Action: --------- 0 of 4 selected

	SLUG	SOURCE	FLAGS
	myorg	local filesystem at ../continuous-ato-kit	
	apps-dev	https://github.com/GovReady/govready-apps-dev	
	system	local filesystem at modules/system	SYSTEM
		null source	

4 app sources

Each App Source points GovReady-Q to a directory or repository of compliance apps.



Add app source

Slug: mygitrepo

A unique URL-safe string that names this AppSource.

How is this AppSource accessed?

Source Type: Git Repository over HTTPS (Public Repository)

What kind of app source is it?

URL: https://github.com/GovReady/govready-apps-dev

The https: URL to a public git repository, e.g. https://github.com/GovReady/govready-q.

The first App Source field is the Slug. The Slug is a short name you assign to the App Source to distinguish it from other App Sources. The Slug is used to form URLs in GovReady-Q's compliance apps catalog, so it may only contain letters, numbers, dashes, underscores, and other URL path-safe characters.

There are four types of App Sources: local directories, remote git repositories using HTTP which are typically public repositories, remote git repositories using SSH which typically use SSH deploy keys for access, and remote GitHub repositories using a GitHub username and password for access.

8.7.1 Local Directory

The Local Directory source type directs GovReady-Q to load compliance apps from a directory on the same machine GovReady-Q is running on. (When deploying with Docker, that's on the container filesystem unless a path has been mounted to a volume or to the host machine.)

In the `Path` field, enter the path to a local directory containing compliance apps. This path is expected to contain a sub-directory for each compliance app contained in this source. For instance, if you have this directory layout:

```
.
└── home
    └── user
        └── compliance_apps
            ├── myfirstapp
            │   └── app.yaml
            └── mysecondapp
                └── app.yaml
```

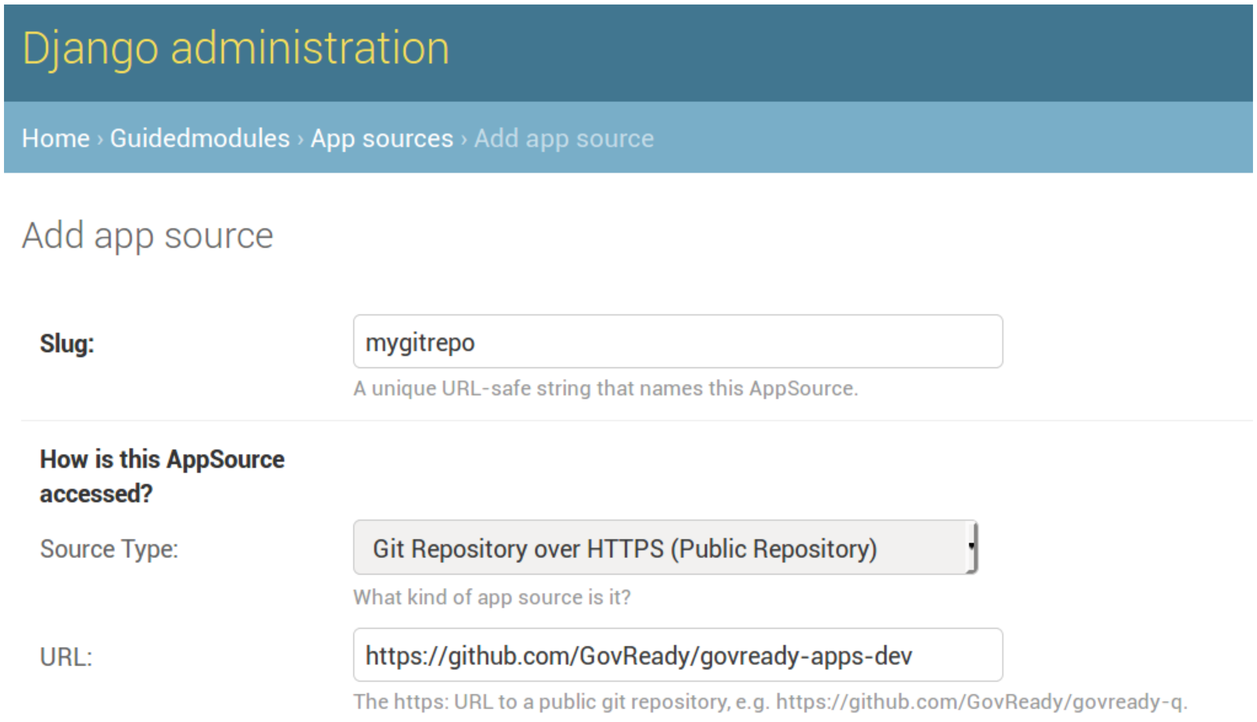
then your Path would be `/home/user/compliance_apps`.

The path can be absolute or relative to the path in which GovReady-Q is installed.

8.7.2 Git Repository over HTTPS

The Git Repository over HTTPS source type is for git repositories, such as on GitHub or GitLab, that can be cloned using an HTTPS URL. These repositories are typically public, or in an enterprise environment public within your organization's network.

Paste the HTTPS git clone URL — such as https://github.com/GovReady/govready-apps-dev — into the URL field. Here's what that looks like:



Add app source

Slug: mygitrepo

A unique URL-safe string that names this AppSource.

How is this AppSource accessed?

Source Type: Git Repository over HTTPS (Public Repository)

What kind of app source is it?

URL: https://github.com/GovReady/govready-apps-dev

The https: URL to a public git repository, e.g. https://github.com/GovReady/govready-q.

The other fields can be left blank.

The `Path` field optionally specifies a sub-directory within the repository in which the compliance apps are stored if they are not stored in the root of the repository. For instance if the repository has a directory layout similar to:

```
.
└── github.com/organization/repository
    └── apps
        ├── myfirstapp
        │   └── app.yaml
        └── mysecondapp
            └── app.yaml
```

then set the `Path` field to `apps`.

If the compliance apps are not in the repository's default branch (i.e. something other than the typical `master` default branch), then set the `Branch` field to the name of the branch to read the compliance apps from.

You can use HTTPS to access private repositories by placing your username and password or personal access token into the URL, such as:

```
https://username:password@github.com/GovReady/govready-apps-dev
```

Since this requires user credentials, it should be avoided for production deployments in favor of using Git Repository over SSH (see below).

8.7.3 Git Repository over SSH

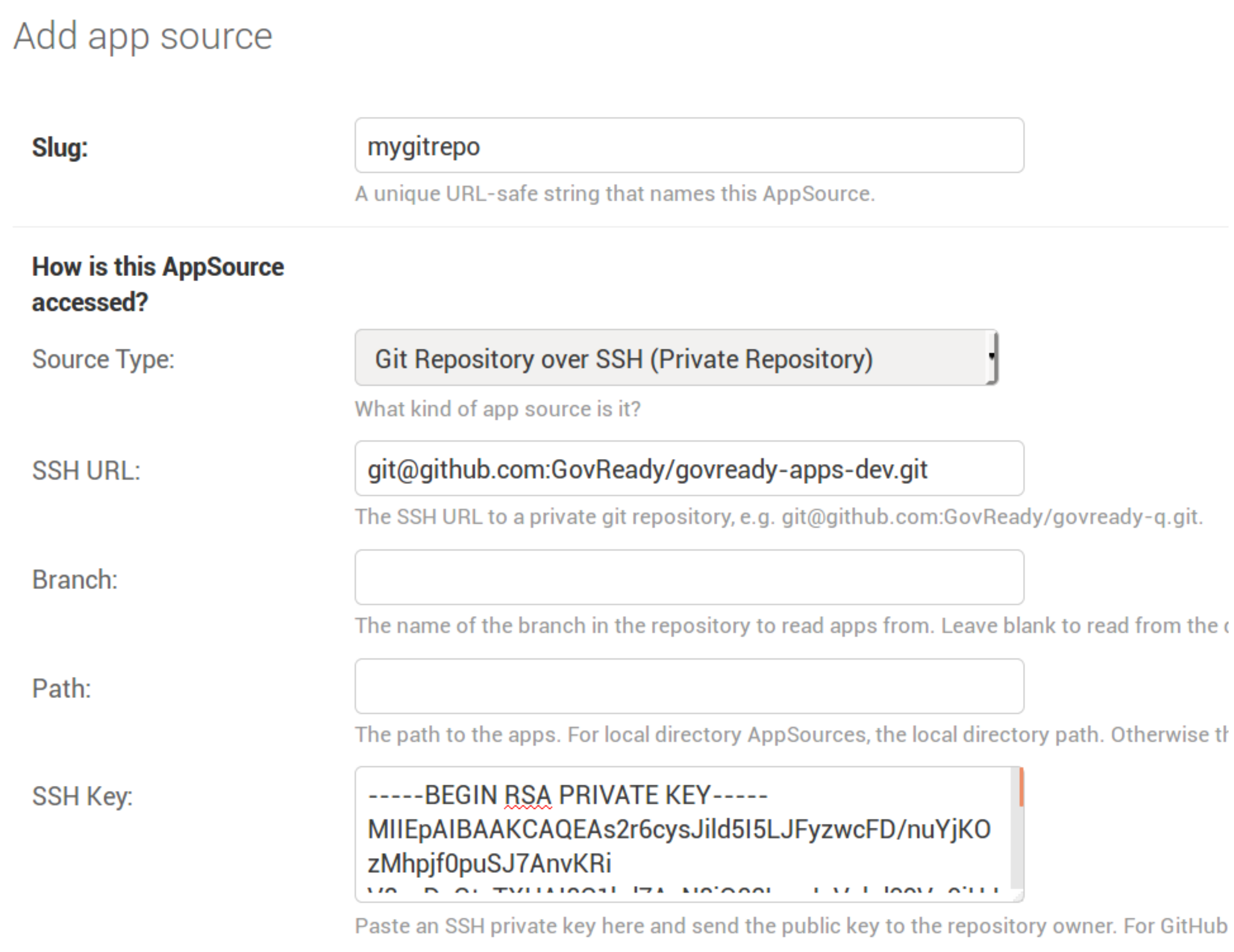
If your git repository is private and accessible using an SSH URL (which typically looks like git@github.com:organization/repository.git) and an SSH public/private keypair, such as with GitHub or GitLab deploy keys, then use the Git Repository over SSH source type.

Create a new SSH key for your GovReady-Q instance to be used as a deploy key:

```
ssh-keygen -q -t rsa -b 2048 -N "" -C "_your-repo-name_-deployment-key" -f ./repo_
↪deploy_key
```

Your GovReady-Q instance will hold the private key half of the newly generated keypair, and your source code control system will hold the public key.

Back in the Django admin, set the Source Type to Git Repository over SSH. Paste the git clone SSH URL into the URL field. Then open the newly generated file `repo_deploy_key` and paste its contents into the SSH Key field. Here's what that looks like:



Add app source

Slug: mygitrepo

A unique URL-safe string that names this AppSource.

How is this AppSource accessed?

Source Type: Git Repository over SSH (Private Repository)

What kind of app source is it?

SSH URL: git@github.com:GovReady/govready-apps-dev.git

The SSH URL to a private git repository, e.g. git@github.com:GovReady/govready-q.git.

Branch:

The name of the branch in the repository to read apps from. Leave blank to read from the

Path:

The path to the apps. For local directory AppSources, the local directory path. Otherwise th

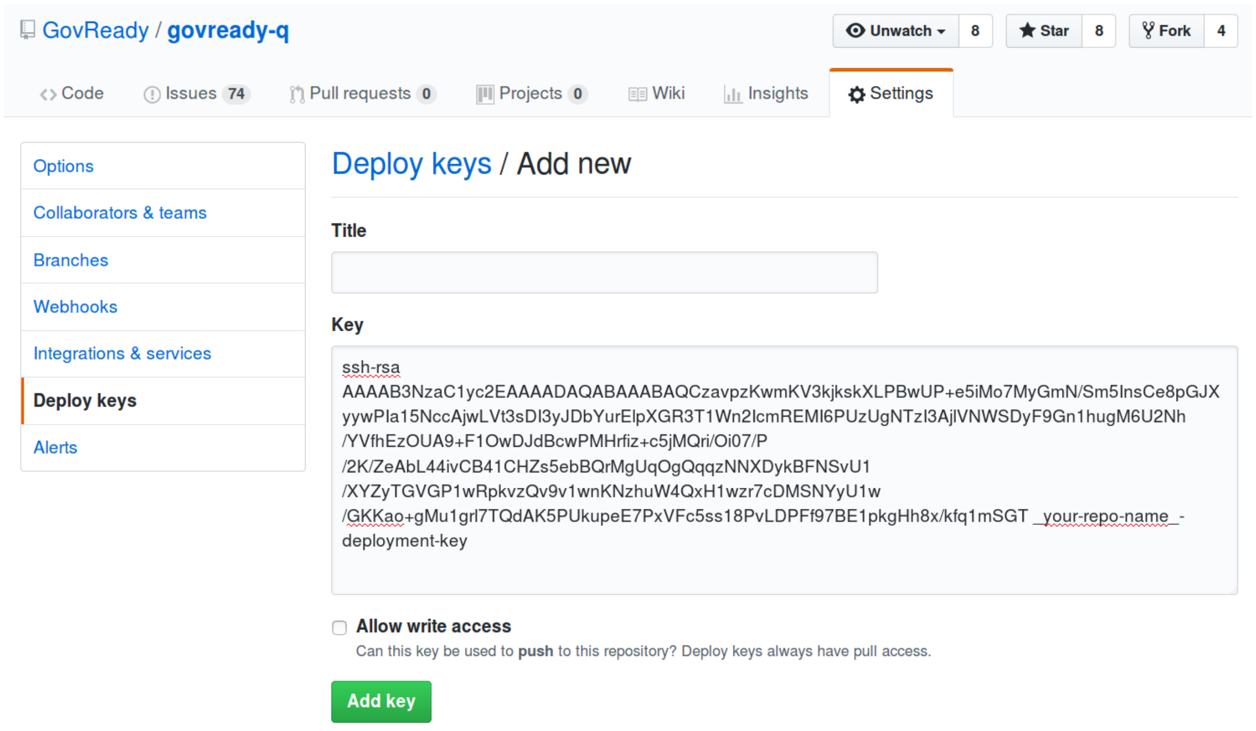
SSH Key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAs2r6cysJild5I5LJFyzwcFD/nuYjKO
zMhpjf0puSJ7AnvKRi
```

Paste an SSH private key here and send the public key to the repository owner. For GitHub

The other fields can be left blank. `Path` and `Branch` can be set the same as with the Git Repository over HTTPS source type (see above).

Copy the public key in the newly generated file `repo_deploy_key.pub` into the deploy keys section of your source code repository. Here is what that looks like on GitHub:



Deploy keys / Add new

Title

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCzavpzKwmKV3kjkskXLPBwUP+e5iMo7MyGmN/Sm5InsCe8pGJX
yywPIa15NccAjwLVt3sDI3yJDbYurElpXGR3T1Wn2IcmREMI6PUzUgNTzI3AjlVNWSDyF9Gn1hugM6U2Nh
/YVfhEzOUA9+F1OwDJdBcwPMHrfiz+c5jMQri/Oi07/P
/2K/ZeAbL44ivCB41CHZs5ebBQrMgUqOgQqqzNNXDykBFNSvU1
/XYZyTGVGP1wRpkvzQv9v1wnKNzhuW4QxH1wzr7cDMSNYyU1w
/GKKao+gMu1grl7TQdAK5PUkupeE7PxVFc5ss18PvLDPFf97BE1pkgHh8x/kfq1mSGT _your-repo-name_-
deployment-key
```

Allow write access

Can this key be used to **push** to this repository? Deploy keys always have pull access.

Make the key read only by leaving "Allow write access" field unchecked and click `Add the key` to save the key.

8.7.4 GitHub Repository using the GitHub API

This source type can be used to access private GitHub repositories using a GitHub username and password or a username and personal access token.

Set the `Repository` field to the organization name and repository name, separated by a slash, as in the repository's URL following `github.com/`. In `Other Parameters`, paste a small YAML-formatted document holding a GitHub username and password or username and personal access token, formatted as follows:

```
auth:
  user: 'myusername'
  pw: 'mypassword'
```

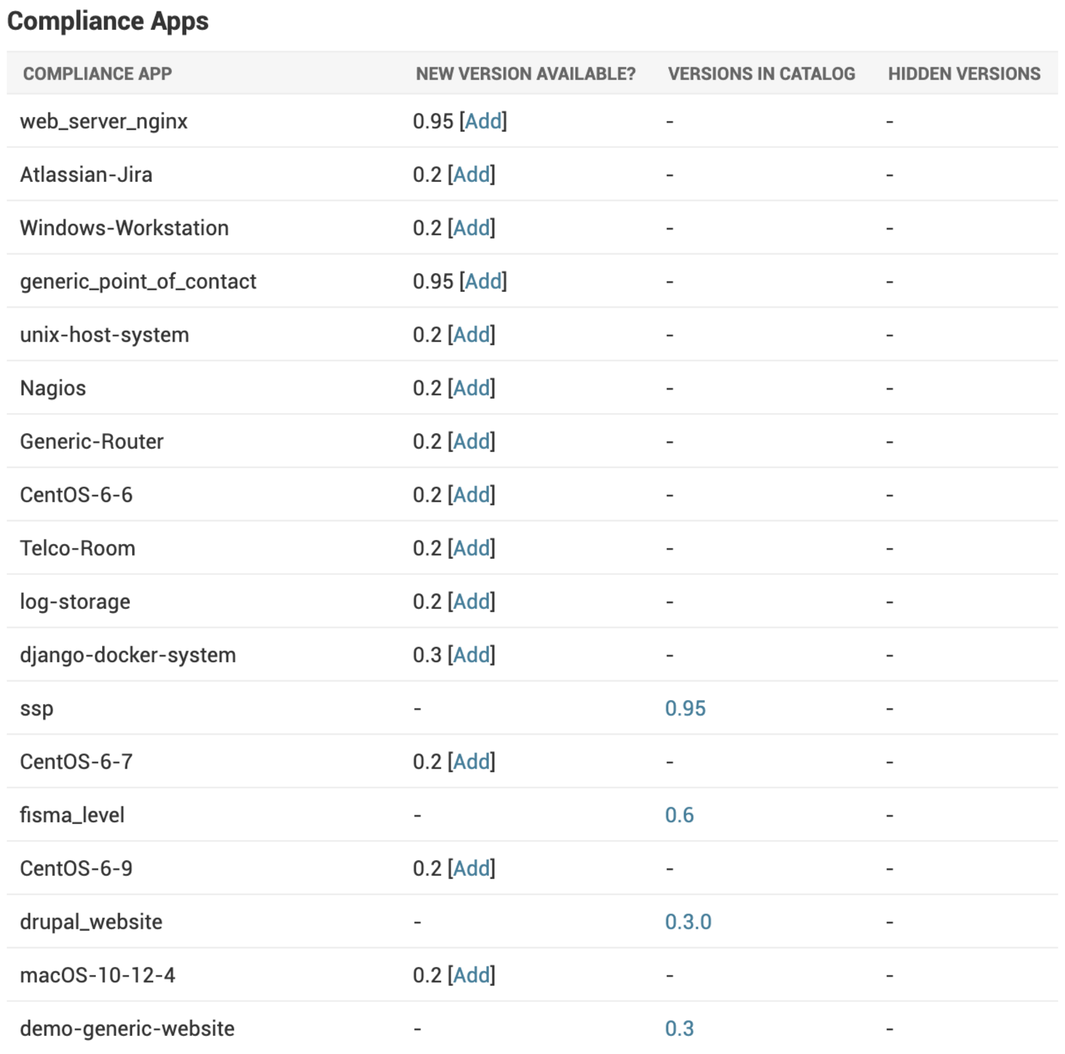
The other fields can be left blank. `Branch` can be set the same as with the Git Repository over HTTPS source type (see above).

Since this source type requires user credentials, it should be avoided for production deployments in favor of using Git Repository over SSH.

Compliance Apps

Compliance Apps are the actual assessments and questionnaires, the "data packs", that drive GovReady-Q.

The AppSource admin now lists all of the apps provided by the source and has links to import new app versions into the database and to see the app versions already in the database by version number. When the App Source is defined, additional options appear on the database App Source admin page to selectively add individual projects and assessments from the App Source repository to the be published on GovReady-Q. Any time the individual admin page for an App Source is viewed, the App Source is rescanned and new versions of the apps are displayed to be selectively added to be available to users on GovReady-Q.



Compliance Apps

COMPLIANCE APP	NEW VERSION AVAILABLE?	VERSIONS IN CATALOG	HIDDEN VERSIONS
web_server_nginx	0.95 [Add]	-	-
Atlassian-Jira	0.2 [Add]	-	-
Windows-Workstation	0.2 [Add]	-	-
generic_point_of_contact	0.95 [Add]	-	-
unix-host-system	0.2 [Add]	-	-
Nagios	0.2 [Add]	-	-
Generic-Router	0.2 [Add]	-	-
CentOS-6-6	0.2 [Add]	-	-
Telco-Room	0.2 [Add]	-	-
log-storage	0.2 [Add]	-	-
django-docker-system	0.3 [Add]	-	-
ssp	-	0.95	-
CentOS-6-7	0.2 [Add]	-	-
fisma_level	-	0.6	-
CentOS-6-9	0.2 [Add]	-	-
drupal_website	-	0.3.0	-
macOS-10-12-4	0.2 [Add]	-	-
demo-generic-website	-	0.3	-

When starting a compliance app (i.e. creating a new project), we no longer have to import the app from the remote repository — instead, we create a new Project and set its root_task to point to a Module in an AppVersion already in the database.

App loading is refactored in a few places. The routines for getting app catalog information from the remote app data are removed since now we only need it for apps already stored in the database.

The AppSource admin's approved app lists form is removed since adding apps into the database is now an administrative function and the database column for it is dropped.

AppVersion now has a boolean field for whether the instance should be included in the compliance apps catalog for users to start new projects with that app.

CHAPTER

NINE

MIGRATION GUIDE FOR GOVREADY-Q V0.8.6 TO V0.9.0

9.1 Major Changes between v0.8.6 and v0.9.0

Changes GovReady-Q users will encounter in v0.9.0:

- Organization subdomains are no longer used - everything is on the main domain.
- Individual Questionnaires/Assessments from an App Source must now be explicitly added to the catalog via the Django admin page. More information can be found in the version 0.9.0 section of the GovReady-Q documentation.
- Assessments are referred to as "Projects" in the UI.
- "Projects" are organized into "Portfolios." Every Project belongs to exactly one Portfolio.
- Users can be added to Portfolios and be granted different permissions in Portfolios via an improved permission model.

v0.8.6	v0.9.0	What migrating does
Subdomains.	Single domain.	Single domain is used as per `host` param in `local/environment.json` file or Docker config params.
Organizations associated with different subdomains.	Single "master" organization.	A "Portfolio" is created for each Organization with the same name as the Subdomain. Organizations continue to exist in database, but are not used.
Users are associated with multiple subdomain organizations.	All users associated with single instance.	Preserves users.
User have different profiles for each subdomain organization.	User has single profile.	First profile is kept and can be edited by user.
User "is staff" and "is super user" set in Django Admin.	User "is staff" and "is super user" set in Django Admin.	Roles preserved, see Roles and Permissions in v0.9.0 section.
Started and completed apps.	Started and completed apps.	All existing apps/questionnaires are preserved. Each questionnaire becomes associated with the "Portfolio" having the same name as the Organization to which the project was previously associated.
User signs in under each subdomain.	User signs in once.	Subdomain associates are removed.

Complete list of changes: https://github.com/GovReady/govready-q/blob/0.9.0.dev/CHANGELOG.md.

9.2 Migration Process

9.2.1 General Guidelines

- Make sure your live data is backed up, and can be found and restored properly, before, during, and after the migration.
- Do test migrations on test servers first, to ensure you understand the process and have worked out any kinks, before working on production servers.
- Check for customizations, and preserve or modify them as needed.

9.2.2 Back Up Your Production Data

- Ensure that you have a backup of your production data, and that it is safe, and will be available for a successful restore.

9.2.3 Read the Documentation

- Familiarize yourself with the migration process before conducting the first test migration.
- Make sure GovReady-Q official documentation is working for you.

9.2.4 Do a Test Migration

- Use test data that will model and exercise the same features as your production data.
- Use a clean install of the version you run in production, with test data installed.
- Migrate your customizations to the test platform.
- Perform the upgrade to v0.9.0.release.
- Test to ensure the upgrade performed properly. Keep notes as you test.
- Repeat until you're comfortable with the process and results.
- Optionally have selected end-users sign into the upgraded test instance to perform their own tests.

9.2.5 Distribute a Migration Plan

- Create a migration plan. Include user impacts, timelines, contingency plans, and technical details (perhaps in a separate technical plan).
- Confirm that your colleagues who are responsible, accountable, consulted, and informed about the migration are satisfied with the plan.
- Distribute the plan to anybody affected.
- When migration starts, communicate to users that migration is starting, and keep a communication line with them open.

9.2.6 Do the Production Migration for Deployments with Source Code

- Take your GovReady-Q instance offline.
- Back up the most recent version of the production database.
- Test a restore of most recent version of the database.
- Update code to version 0.9.0.release.
- Run `pip install -r requirements.txt` to get new libraries.
- Update any template and file customizations in your workstream.
- Run `python manage.py migrate` to update your database schema.
- Run `python manage.py runserver` to run your updated instance.

9.2.7 Do the Production Migration for Deployments with Docker/Containers

- Take your GovReady-Q instance offline.
- Back up the most recent version of the production database.
- Test a restore of most recent version of the database.
- Synchronize your container customizations to produce a new version of your container.
- Deploy container running version 0.9.0 **with environment variable ''DB_BACKED_UP_DO_UPGRADE'' set to "True"**. (This special environment variable is required to avoid accidental running of database migrations before database has been backed up.)
- Docker will automatically run migrations as part of deployment.

9.2.8 Migration Finalization and Testing

- Test the new instance.
- Back up the newly migrated production database.

9.2.9 Post Migration Clean Up

- Mark all old Notifications `emailed` as True. (v0.9.0 notification checks emailed status of notifications and sets emailed to True after email is sent.)
- Review Help Squad, Reviewers, and Administrators.
  - The migration converts Organizations to Portfolios.
  - Help Squad, Reviewers, and Organization admins are only preserved on the "main" organization (e.g., first organization created).
  - The migration does not modify the Help Squad or Reviewers.
  - Organization Admins are associated with the "main" organization.
- You should review who has these permissions and adjust accordingly after migrating.
- Release the production instance to users.

9.3 Roles and Permissions in v0.9.0

Permission/Role Name	Description	What Happens During Migration
Organization > get_who_can_read	A user can see an Organization if: they have read permission on any Project within the Organization, they are an editor of a Task within a Project within the Organization (but might not otherwise be a Project member), they are a guest in any Discussion on TaskQuestion in a Task in a Project in the Organization.	A Portfolio is created for every organization that exists.
project-membership	See Project > has_read_priv	See below.
Project > has_read_priv (Inverse is Project > get_all_participants)	Team members + anyone with read privs to a task within this project + anyone who is a guest in discussion within this project.	See below.
Project > is_admin	Person flagged as project admin in ProjectMembership.	Grant project_delete permission on project and portfolio_owner permission for portfolio for which project is a part
Project > is_member Project > editor_of task(s)Project > discussion_guest_in discussion(s)	Various permissions in a project.	Grant project_view, project_add, project_change, for project of which project is a part; Grant Portfolio_view, portfolio_add, portfolio_change for which project is a part after migration.
task_editor	The user that has primary responsibility for completing this Task.	Grant project_view, project_add, project_change, for project of which task is a part; Grant Portfolio_view, portfolio_add, portfolio_change for which task is a part after migration.
help_squad	Receives all discussion messages	Not modified by migration
reviewer	Can set review status of answers.	Not modified by migration
superuser	Django designated superuser.	Not modified by migration
Folder permissions	A folder object exists but is not used.	Not modified by migration
portfolio_owner	Permission on portfolio object, can invite others to portfolio and can make others to portfolio owner	If user was project_membership and had project_membership admin flag True, user is made portfolio owner.
Portfolio_view, portfolio_add, portfolio_delete, portfolio_change	Permissions on Portfolio objects. Currently everyone who has one of these has all of these permissions.	Every user gets a portfolio with their name for which user is the portfolio_owner. For every organization, a portfolio is created with the same name and associating the organization projects with the portfolio. Users get access to the projects.
project_view, project_add, project_change, project_delete	Permissions on Project objects. Currently everyone who has one of these has all of these permissions.	If user has project_membership on project, user gets project_view, project_add, project_change. If user has project_membership on project and project_membership admin flag True, user also gets project_delete. If user has task object for a project, user gets project_view, project_add, project_change,

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search