Django apps that run on Google Kubernetes Engine (GKE) (/kubernetes-engine/) scale well because they run on the same infrastructure that powers all of Google's products.

This tutorial assumes you are familiar with Django web development. If you are new to Django development, it's a good idea to work through writing your first Django app (https://docs.djangoproject.com/en/1.9/intro/tutorial01/) before continuing. In that tutorial, the app's models represent polls that contain questions, and you can interact with the models by using the Django admin console.

This tutorial requires Python 2.7 or 3.4 or later (https://www.python.org/). You also need to have Docker installed (https://docs.docker.com/engine/installation/).

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).
2. In the Cloud Console, on the project selector page, select or create a Cloud project.

   Note: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

   Go to the project selector page (https://console.cloud.google.com/projectselector2/home/dashboard)
3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (/billing/docs/how-to/modify-project).
4. Install and initialize the Cloud SDK (/sdk/docs/).
5. Enable the Datastore, Pub/Sub, Cloud Storage JSON, Stackdriver Logging, and Google+ APIs.

   Enable the APIs (https://console.cloud.google.com/flows/enableapi?apiid=datastore.googleapis.com,pubsub,storage_api,logging,plus)

After you've completed the prerequisites, download and run the Django sample app. The following sections guide you through configuring, running, and deploying the app.

The code for the Django sample app is in the `GoogleCloudPlatform/python-docs-samples` (https://github.com/GoogleCloudPlatform/python-docs-samples) repository on GitHub.

1. You can either download the sample (https://github.com/GoogleCloudPlatform/python-docs-samples/archive/master.zip) as a zip file and extract it or clone the repository to your local machine by using the following command:

2. Change to the directory that contains the sample code:

When deployed, your app uses the Cloud SQL Proxy that is built in to the App Engine environment to communicate with your Cloud SQL instance. However, to test your app locally, you must install and use a local copy of the proxy in your development environment.

Learn more about the Cloud SQL Proxy (/sql/docs/postgres/sql-proxy).

To perform basic admin tasks on your Cloud SQL instance, you can use the PostgreSQL client.

You must authenticate `gcloud` (/sql/docs/postgres/sql-proxy#gcloud) before you can use the Cloud SQL Proxy to connect from your local mac

Before using Cloud SQL, you must enable the Cloud SQL Admin API:

Download and install the Cloud SQL Proxy. The Cloud SQL Proxy connects to your Cloud SQL instance when running locally.

If your operating system isn't included here, you can also compile the proxy from source (http://github.com/GoogleCloudPlatform/cloudsql-proxy).

1. Create a Cloud SQL for PostgreSQL instance. (/sql/docs/postgres/create-instance)

   Name the instance `polls-instance` or similar. It can take a few minutes for the instance to be ready. When the instance is ready, it's visible in the instances list.

2. Use the Cloud SDK to run the following command where `[YOUR_INSTANCE_NAME]` represents the name of your Cloud SQL instance:

   In the output, note the value shown for `[CONNECTION_NAME]`.

   The `[CONNECTION_NAME]` value is in the format `[PROJECT_NAME]:[REGION_NAME]:[INSTANCE_NAME]`.

1. Start the Cloud SQL Proxy by using the `[CONNECTION_NAME]` value from the previous step:

   Replace `[YOUR_INSTANCE_CONNECTION_NAME]` with the `[CONNECTION_NAME]` value that you recorded in the previous step.

   This step establishes a connection from your local computer to your Cloud SQL instance for local testing purposes. Keep the Cloud SQL Proxy running the entire time you test your app locally.

2. Create a Cloud SQL user and database:

The proxy requires a service account with Editor privileges for your Cloud SQL instance. For more information about service accounts, see the Google Cloud authentication overview (/docs/authentication/).

To create a service account with the required permissions, you must have `resourcemanager.projects.setIamPolicy` permission. This ssion is included in the Project Owner, Project IAM Admin, and Organization Administrator roles.
ust also have enabled the Cloud SQL Admin API.

When you use a service account to provide the credentials for the proxy, you must create it with sufficient permissions. If you are using the finer-grained Identity Access and Management (IAM) roles to manage your Cloud SQL permissions, you must give the service account a role that includes the cloudsql.instances.connect permission. The predefined Cloud SQL roles that include this permission are:

- Cloud SQL Client
- Cloud SQL Editor

- Cloud SQL Admin

If you are using the legacy project roles (Viewer, Editor, Owner), the service account must have at least the Editor role.

1. Go to the **Service accounts** page of the Google Cloud Console.

   Go to the Service accounts page (https://console.cloud.google.com/iam-admin/serviceaccounts/)

2. Select the project that contains your Cloud SQL instance.
3. Click **Create service account**.
4. In the **Create service account** dialog, provide a descriptive name for the service account.
5. For **Role**, select one of the following roles:
   - **Cloud SQL > Cloud SQL Client**
   - **Cloud SQL > Cloud SQL Editor**
   - **Cloud SQL > Cloud SQL Admin**
6. Change the **Service account ID** to a unique, easily recognizable value.
7. Click **Furnish a new private key** and confirm that the key type is `JSON`.
8. Click **Create**.

   The private key file is downloaded to your machine. You can move it to another location. Keep the key file secure.

Use the following commands to set environment variables for database access. These environment variables are used for local testing.

1. This application is represented in a single Kubernetes configuration, called `polls`. In `polls.yaml` replace `<your-project-id>` with your Google Cloud project ID.
2. Run the following command and note the value of `connectionName`:

3. In the `polls.yaml` file, replace `<your-cloudsql-connection-string>` with the `connectionName` value.

1. To run the Django app on your local computer, set up a Python development environment (/python/setup), including Python, pip, and virtualenv.
2. Create an isolated Python environment and install dependencies. If your Python 3 installation has a different name, use that in the first command:

3. Run the Django migrations to set up your models:

4. Start a local web server:

5. In your browser, go to http://localhost:8000 (http://localhost:8000).

   You see a page with the following text: "Hello, world. You're at the polls index." The Django web server running on your computer delivers the sample app pages.
6. Press `Control+C` to stop the local web server.

1. Create a superuser. You need to specify a username and password.

2. Run the main program:

3. In your browser, go to http://localhost:8000/admin (http://localhost:8000/admin).
4. Log in to the admin site using the username and password you used when you ran `createsuperuser`.

When the app is deployed to Google Cloud, it uses the Gunicorn server. Gunicorn doesn't serve static content, so the app uses Cloud Storage to serve static content.

1. Create a Cloud Storage bucket and make it publicly readable. Replace `[YOUR_GCS_BUCKET]` with a bucket name of your choice. For example, you could use your project ID as a bucket name.

2. Gather all the static content locally into one folder:

3. Upload the static content to Cloud Storage:

4. In `mysite/settings.py`, set the value of `STATIC_URL` to the following URL, replacing `[YOUR_GCS_BUCKET]` with your bucket name:

1. To initialize GKE, go to the **Clusters** page.

   Go to the Clusters page (https://console.cloud.google.com/kubernetes/list)

   When you use GKE for the first time in a project, you need to wait for the "Kubernetes Engine is getting ready. This may take a minute or more" message to disappear.

2. Create a GKE cluster:

   Did you get the error: "Project [PROJECT_ID] is not fully initialized with the default service accounts." (#create-cluster)?

3. After the cluster is created, use the `kubectl` command-line tool, which is integrated with the `gcloud` tool, to interact with your GKE cluster. Because `gcloud` and `kubectl` are separate tools, make sure `kubectl` is configured to interact with the right cluster.

1. You need several secrets (https://kubernetes.io/docs/concepts/configuration/secret/) to enable your GKE app to connect with your Cloud SQL instance. One is required for instance-level access (connection), while the other two are required for database access. For more information about the two levels of access control, see Instance access control (/sql/docs/mysql/instance-access-control).

   a. To create the secret for instance-level access, provide the location (`[PATH_TO_CREDENTIAL_FILE]`) of the JSON service account key you downloaded when you created your service account (see Creating a service account (#creating_a_service_account)):

   b. To create the secrets for database access, use the SQL `[PROXY_USERNAME]` and `[PASSWORD]` defined in step 2 of Initializing your Cloud SQL instance (#initializing_your_cloud_sql_instance):

2. Retrieve the public Docker image for the Cloud SQL proxy.

3. Build a Docker image, replacing `<your-project-id>` with your project ID.

4. Configure Docker to use `gcloud` as a credential helper, so that you can push the image to Container Registry (/container-registry/):

5. Push the Docker image. Replace `<your-project-id>` with your project ID.

   Note: This command requires write access to Cloud Storage. If you run this tutorial on a Compute Engine instance, your access to Cloud Storage might be read-only. To get write access, create a service account (https://console.cloud.google.com/project/_/iam-admin/serviceaccounts) and use the service account to authenticate (/sdk/gcloud/reference/auth/activate-service-account) on your instance.

6. Create the GKE resource:

After the resources are created, there are three `polls` pods on the cluster. Check the status of your pods:

Wait a few minutes for the pod statuses to display as `Running`. If the pods aren't ready or if you see restarts, you can get the logs for a particular pod to figure out the issue. `[YOUR-POD-ID]` is a part of the output returned by the previous `kubectl get pods` command.

After the pods are ready, you can get the public IP address of the load balancer:

Go to the `EXTERNAL-IP` address in your browser to see the Django basic landing page and access the admin console.

The Django sample app was created using the standard Django tooling. These commands create the project and the polls app:

The `settings.py` contains the configuration for your SQL database:

kubernetes_engine/django_tutorial/mysite/settings.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/kubernetes_engine/django_tutorial/mysite/settings.py)

itHub (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/kubernetes_engine/django_tutorial/mysite/settings.py)

The `polls.yaml` file specifies two Kubernetes resources. The first is the Service (http://kubernetes.io/docs/user-guide/services/), which defines a consistent name and private IP address for the Django web app. The second is an HTTP load balancer (/kubernetes-engine/docs/load-balancer) with a public-facing external IP address.

kubernetes_engine/django_tutorial/polls.yaml (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/kubernetes_engine/django_tutorial/polls.yaml)

iew on GitHub (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/kubernetes_engine/django_tutorial/polls.yaml)

The service provides a network name and IP address, and GKE pods run the app's code behind the service. The `polls.yaml` file specifies a deployment (http://kubernetes.io/docs/user-guide/deployments/) that provides declarative updates for GKE pods. The service directs traffic to the deployment by matching the service's selector to the deployment's label. In this case, the selector `polls` is matched to the label `polls`.

kubernetes_engine/django_tutorial/polls.yaml
(https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/kubernetes_engine/django_tutorial/polls.yaml)

iew on GitHub (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/kubernetes_engine/django_tutorial/polls.yaml)