To do this la you will need to setup a temp. Google Cloud Service account. Please, ask your instructor if you need help doing so.

Automating Image Building With Kubernetes

In this tutorial you will deploy a fully-functional implementation of the automated image building pipeline. You will use Google Container Engine and Kubernetes to deploy the environment.

If you follow these instructions exactly, you will deploy (2) g1-small GCE instances and a network load balancer, all resources that are billed for. It is very important that you follow the instructions to turn down the cluster if you do not want to continue to be billed for these resources.

Be sure to create a brand new project for this tutorial (instructions are in the deploy sections below). Also be sure to complete the Delete the Deployment section when you’re done. It’s super quick and will tear down everything you created.

Deploy

Deployment Requirements - Before you deploy the sample you'll need to make sure a few things are in order:

1. Create a new project in the Google Developer Console and note the new project's ID.

2. In the APIs & Auth section of the Google Developers Console of your new project, enable the following APIs:

    - Google Compute Engine
    - Google Container Engine API

3. Install the Cloud SDK verssion 0.9.68 or greater using these instructions.

4. Authenticate to gcloud:

```
$ gcloud auth login
```

5. Set your project:

```
$ gcloud config set project YOUR_PROJECT_ID
```

6. Enable kubernetes features:

```
$ gcloud components update kubectl
```

Quick Deploy

These quick deploy instructions are easiest way to get started. The work to create a Google Container Engine cluster and launch the necessary Kubernetes resources is captured in the `cluster_up.sh` script.

To quick deploy the image builder application:

1. Clone this repository (`$ git clone https://github.com/GoogleCloudPlatform/kube-jenkins-imager.git`) or download and unzip a copy from releases.

2. Navigate to the directory:

```
$ cd kube-jenkins-imager
```

3. From a terminal in the directory you cloned or unzipped, run:

```
$ ./cluster_up.sh
```

The script will take several minutes to complete. The abbreviated output should look similar to:

```
Creating cluster imager...done.
...
...
```

```
<TRUNCATED>
...
...
All resources deployed.
```

4. Continue to the Access Jenkins section (skip the Stepwise Deploy section)

[OPTIONAL] Stepwise Deploy-You can find open the `cluster_up.sh` script and execute the commands from each line for a closer look at everything that’s taking place.

Access Jenkins

1. Access the URL output when you created your deployment. Click the login button and use the username and password that was output by the `cluster_up.sh` script.

2. After a successful login you should see the Jenkins admin landing page with a default backup job running.

Create Credentials

1. Optional: Configure a Jenkins login (in addition to the basic access authentication at the reverse proxy) by navigating to Manage Jenkins >> Configure Global Security and configuring authentication and authorization settings to your requirements.

2. Create a credential by clicking on the Credentials link in the left nav, then clicking the Global credentials link.

3. Click Add Credentials in the left nav, choose Google Service Account from metadata in the Kind dropdown, and click OK. The Project Name will be auto-populated.

Create and Run a Build Job

In the following sections you will clone an existing repo (from the previous Scalable and Resilient Web Applications tutorial that includes a working build configuration. You will then push that repo to your project’s Cloud Repository, create a Jenkins job to build it, and run the job.

Replicate a GitHub Repo

1. Clone the existing sample repository to your workstation (you must have git installed) and go into the new directory:

```
$ git clone https://github.com/GoogleCloudPlatform/scalable-resilient-web-app.git
$ cd scalable-resilient-web-app
```

2. In the Google Developer Console, navigate to Source Code > Browse, click “Get started” then choose “Push code from a local Git repository to your Cloud Repository”, and follow all of

the instructions to push the scalable-resilient-web-app to your Cloud Repository.

3. Click the Browse menu item again and confirm your files are there

4. After you’ve pushed your files to the Cloud Repository, find and copy its Fetch URL for use in the next section:

```
$ git remote -v show -n google | grep Fetch
```

Output:

```
  Fetch URL: https://source.developers.google.com/p/your-new-project/
```

> *NOTE: The URL should be your project ID appended to the string https://source.developers.google.com/p/*

Create Jenkins Job

1. Access Jenkins in your browser. If you don’t remember the URL, you can run the following command in the terminal where you created the deployment to find it:

```
$ echo http://$(kubectl get ingress jenkins --namespace jenkins -o "jsonpath={.status.loadBalancer.ingress[0].ip}"
)
```

2. From the Jenkins main page, choose *New Item, name the item `redmine-immutable-image`, choose Freestyle project, then click OK. It is important the name does not include spaces.

3. Check Restrict where this project can be run and use gcp-packer as the label expression to ensure the job runs on the correct build agents.

4. Under Source Code Management, choose Git, paste your Cloud Repository URL (`https://source.developers.google.com/p/your-project-id`) from the previous section, and choose the credential you created earlier from the dropdown.

5. Under Build Triggers, choose Poll SCM and enter a value for Schedule. In this example, H/5 * * * * will poll the repository every 5 minutes. Choose a value that you consider appropriate.

6. Under Build, click the Add build step dropdown and select Execute shell.

7. Paste the following command. When it runs, it will retrieve the project ID your Jenkins installation is running in, execute Packer to build GCE and Docker images, then push the Docker image to Google Container Registry and finally remove the

local copy of the Docker image.

```

#!/bin/bash
set -e

# Get current project
PROJECT_ID=$(curl -s 'http://metadata/computeMetadata/v1/
project/project-id' -H 'Metadata-Flavor: Google')

# Install packer
curl -L https://releases.hashicorp.com/packer/0.8.6/packe
r_0.8.6_linux_amd64.zip -o /tmp/packer.zip; unzip /tmp/pa
cker.zip -d /usr/local/bin

# Do packer build
packer build \
  -var "project_id=${PROJECT_ID}" \
  -var "git_commit=${GIT_COMMIT:0:7}" \
  -var "git_branch=${GIT_BRANCH#*/}" \
  packer.json

# Create and push Docker version of the image
IMAGE_TAG=gcr.io/${PROJECT_ID}/redmine:${GIT_BRANCH#*/}-$
{GIT_COMMIT:0:7}

# Build image
docker build -t $IMAGE_TAG .
```

```

# Push image
gcloud docker push $IMAGE_TAG

# Remove local image
docker rmi $IMAGE_TAG
```

8. Click Save to save your job!

Run the Build

1. After saving the project, choose the Build Now menu item, then click the job number when it appears.

2. Choose the Console Output menu item and observe the job’s progress.

Packer parallelizes the GCE and Docker builds. You can expect the build to take about 20 minutes; the sample build is updating the OS, building and installing Ruby, and installing the Redmine project management application and all of its gem dependencies.

1. The build is done when you see a Finished: SUCCESS line in the output. A few lines before that you should see the outputs (GCE and Docker iamges) of the build:

```
==> Builds finished. The artifacts of successful builds a
re:
```

```
--> googlecompute: A disk image was created: redmine-1431
028076-master-c84d21f
--> docker: Imported Docker image: 0717053a7fce3c637a5bfd
887954f41b4327e80493eb6492277e2dbb132c2bf4
--> docker: Imported Docker image: gcr.io/your-new-projec
t/redmine:master-c84d21f
...
...
Finished: SUCCESS
```

2. In the Google Developers Console navigate to Compute > Images and confirm that your GCE image for Redmine is there:

Configure Backup

In this section you will configure Jenkins to backup your job configurations and history to Google Cloud Storage.

1. Create a bucket to store the backups in. Copy the output of the command (sample output included below):

```
$ gsutil mb gs://jenkins-backups-$RANDOM-$(date +%s)
```

Output:

```
Creating gs://jenkins-backups-21885-1430974383/...
```

2. From the Jenkins main page, click the jenkins-gcs-backup job,

then choose the Configure menu item.

3. **[OPTION]** Adjust the build schedule to fit your needs.

4. In the Post-build Actions section, ensure your credential is selected, then edit the Storage Location field, replacing the `YOUR_GCS_BUCKET_NAME` string with the name of the bucket you created in the previous step. In the example output above, that would be `jenkins-backups-21885-1430974383`. Save your changes!

5. Click the Build Now menu item to run the backup. It should completely quickly, usually in just a few seconds. The blue orb indicates a successful backup.

6. View the backup in the Google Developers Console by choosing **Storage > Cloud Storage > Storage browser** in the left menu, then clicking your backup bucket in the list, and clicking into the `jenkins-backups` folder. You should see both the date-stamped and LATEST backups.

Practice Restoring a Backup

Now that you have a Jenkins backup, you can use Kubernetes and Google Container Engine to practice restoring the backup. Here's an overview of the process, with code you can execute to complete it.

1. Create a new Replication Controller file for the leader and open

it in your favorite text editor:

```
$ cp leader.yaml leader-restore.yaml
$ vim leader-restore.yaml
```

2. Add an environment variable to the pod spec pointing to the backup and rename the controller (look for the two # MODIFY tokens in the code below to see what you need to change in your file):

```
---
kind: ReplicationController
apiVersion: v1beta3
metadata:
  # MODIFY NAME
  name: jenkins-leader-restored
  labels:
    name: jenkins
    role: leader
spec:
  replicas: 1
  selector:
    name: jenkins
    role: leader
  template:
    metadata:
      name: jenkins-leader
      labels:
        name: jenkins
        role: leader
    spec:
      containers:
      - name: jenkins
        image: gcr.io/cloud-solutions-images/jenkins-gcp-
leader:latest
        command:
        - /usr/local/bin/start.sh
        env:
        - name: GCS_RESTORE_URL
          # MODIFY VALUE
          value: gs://jenkins-backups-21885-1430974383/je
nkins-backups/LATEST.tar.gz
        ports:
        - name: jenkins-http
          containerPort: 8080
        - name: jenkins-discovery
          containerPort: 50000
```

3. Create the new Replication Controller:

```
$ kubectl create -f leader-restore.yaml
```

4. Resize the old leader Replication Controller to 0:

```
$ kubectl resize --replicas=0 replicationcontroller jenki
ns-leader
```

5. Delete the old Replication Controller and rename the new file:

```
$ kubectl delete -f leader.yaml
$ mv leader-restore.yaml leader.yaml
```

6. Refresh the Jenkins URL in your browser until the restored version is available. You should notice your jobs and job history restored.

Delete the Deployment

It is very important (as mentioned in the Very Important Things section of this document) that you delete your deployment when you are done. You will be charged for any running resources.

Whether you followed the Quick Deploy or Stepwise Deploy instructions, deleting resources is very easy. Simply delete the project you created at the beginning of this tutorial:

Navigate to the Projects page of the Google Developer Console, find your project, click the trash can icon to delete, then type the project ID and click Delete Project.