Sunnyvale
A computer sciences company

FullStackConf 2019

Microservices DevOps on Google Cloud Platform

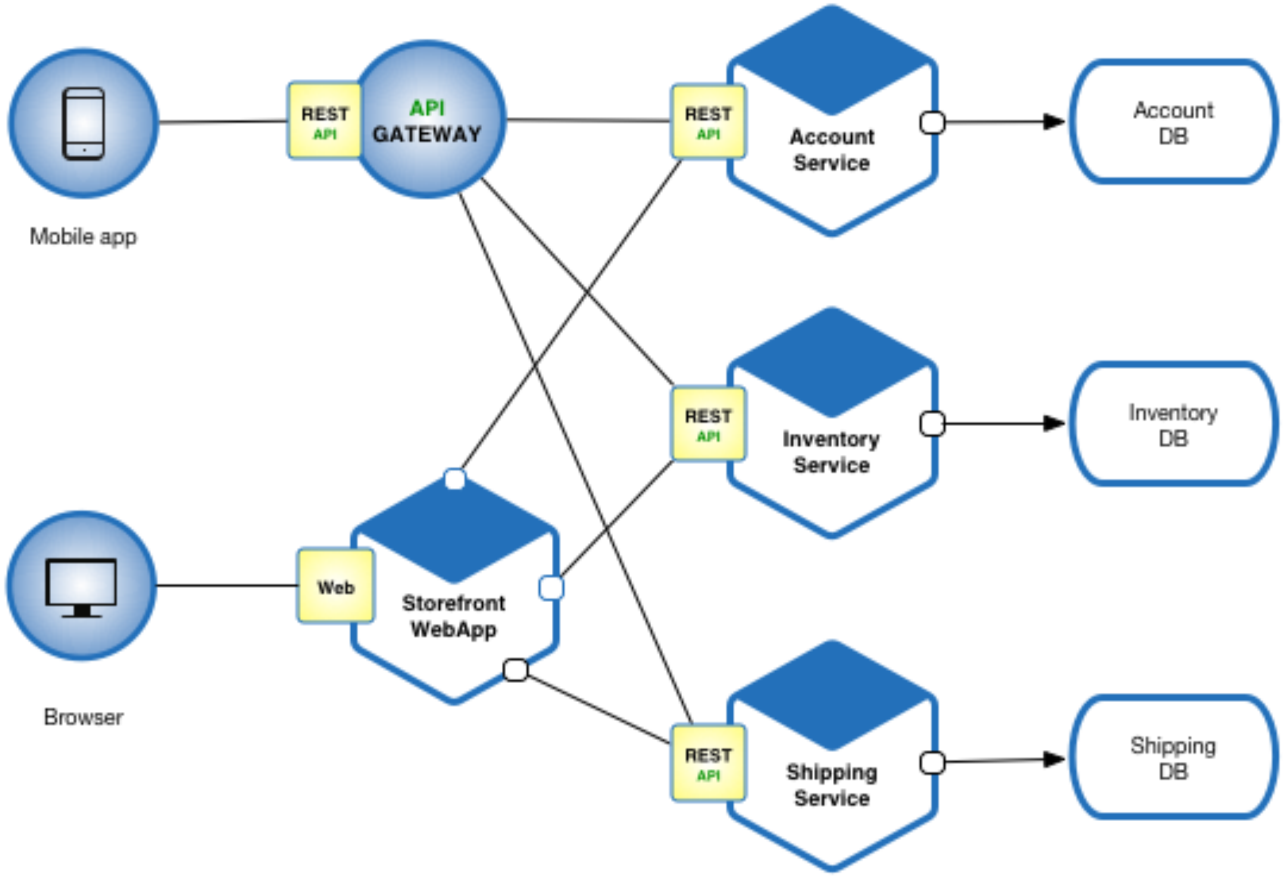
Denis Maggiorotto
Sunnyvale S.r.l.

Turin
10/10/2019

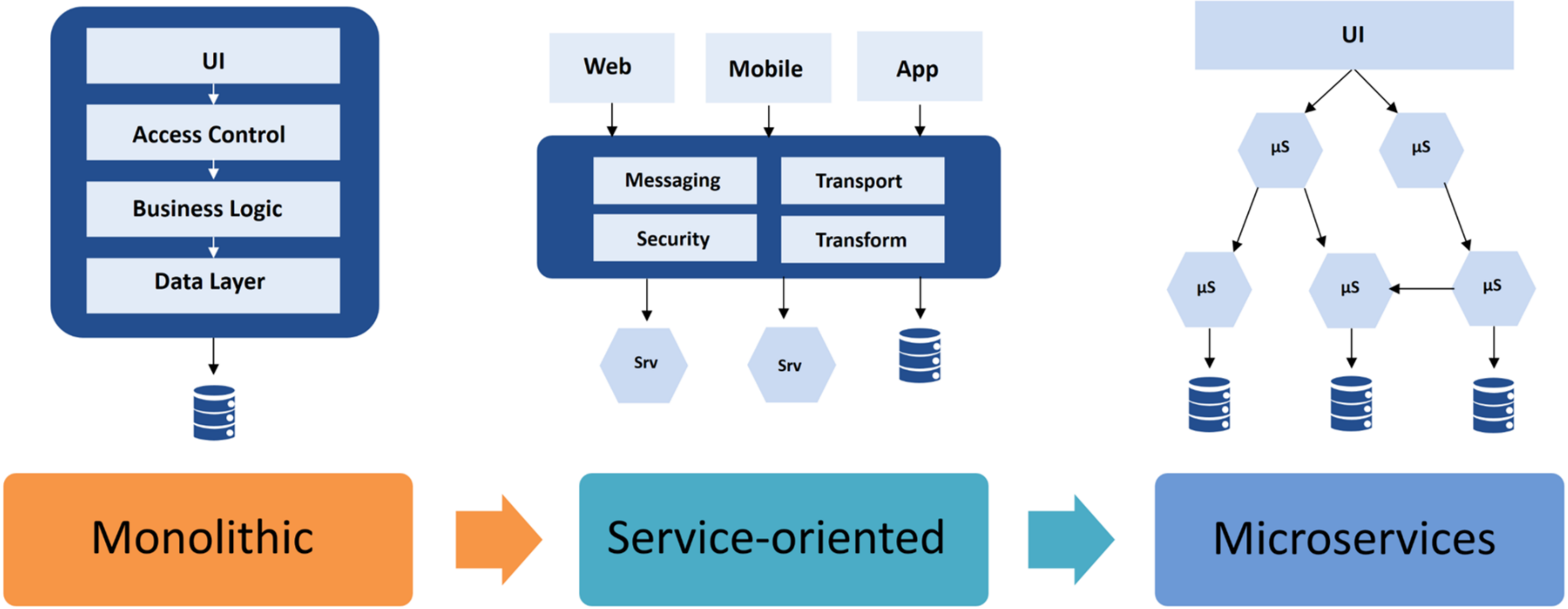
What are microservices?

Microservices - also known as the **microservice architecture** - is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team



Comparing architectural styles



UI
Access Control
Business Logic
Data Layer

Web
Mobile
App
Messaging
Transport
Security
Transform
Srv
Srv

UI
µS
µS
µS
µS
µS

Monolithic

Service-oriented

Microservices

Microservices Pros and cons

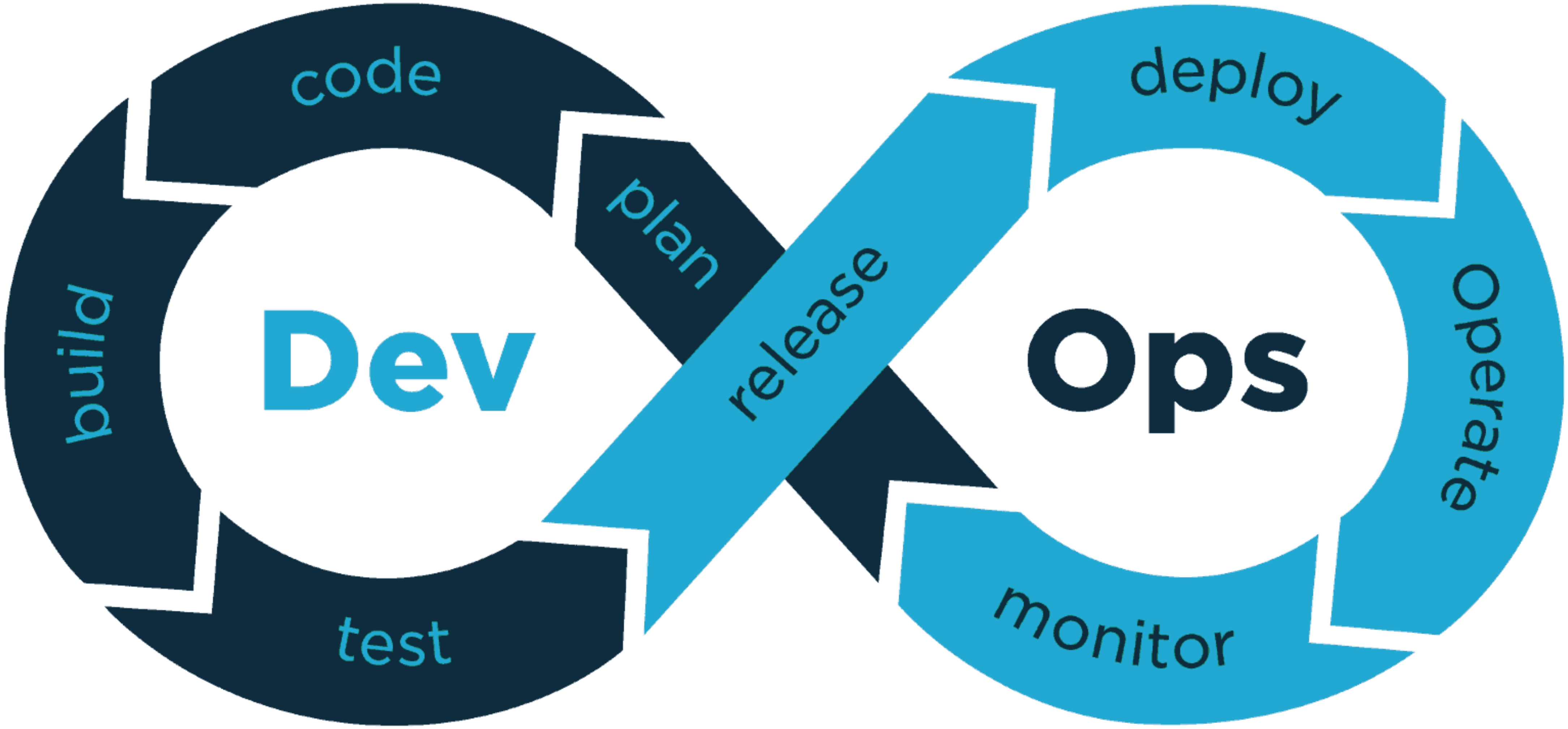
Pros

- Enables the continuous delivery and deployment of large, complex applications
- Each microservice is relatively small (so teams are)
- Improved fault isolation
- Eliminates any long-term commitment to a technology stack

Cons

- Developers must deal with the additional complexity of creating a distributed system
- Deployment complexity
- (Sometimes) increased memory consumption. The microservices architecture replaces N monolithic application instances with NxM services instances

DevOps culture



Keep C.A.L.M.S. and do DevOps

CALMS is a conceptual framework for the integration of development and operations (DevOps) teams, functions and systems within an organization.
The acronym CALMS is credited to the authors of "The DevOps Handbook.” after the first US based Devopsdays in Mountainview 2010.

Culture: there is a culture of shared responsibility

Automation: team members seek out ways to automate as many tasks as possible and are comfortable with the idea of continuous delivery

Lean: team members are able to visualize work in progress (WIP), limit batch sizes and manage queue lengths

Measurement: data is collected on everything and there are mechanisms in place that provide visibility into all systems

Sharing: (a.k.a. Collaboration) there are user-friendly communication channels that encourage ongoing communication between development and operations

Cloud and Infrastructure as Code

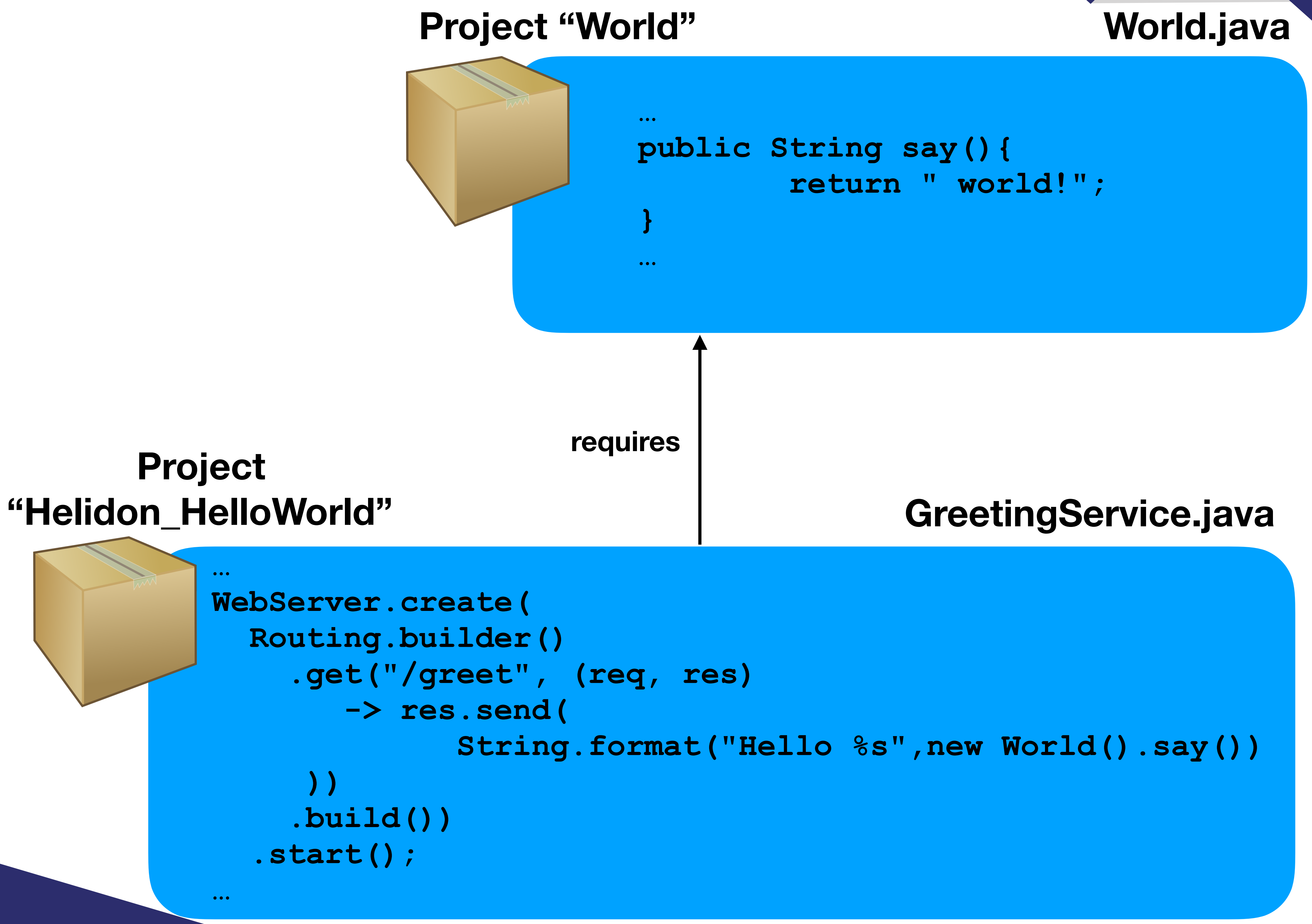
Cloud resources are provisioned with **Infrastructure as Code (IaC)** paradigm.

Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

The value of IaC can be broken down into three measurable categories: cost (reduction), speed (faster execution) and risk (remove errors and security violations).

Terraform is the industry standard solution about IaC in Cloud environment (OpenSource)

The usecase scenario



Project “World”

World.java

```
...
public String say(){
        return " world!";
}
...
```

requires

Project “Helidon_HelloWorld”

GreetingService.java

```
...
WebServer.create(
  Routing.builder()
     .get("/greet", (req, res)
        -> res.send(
              String.format("Hello %s",new World().say())
     ))
     .build())
  .start();
...
```

Components lifecycle

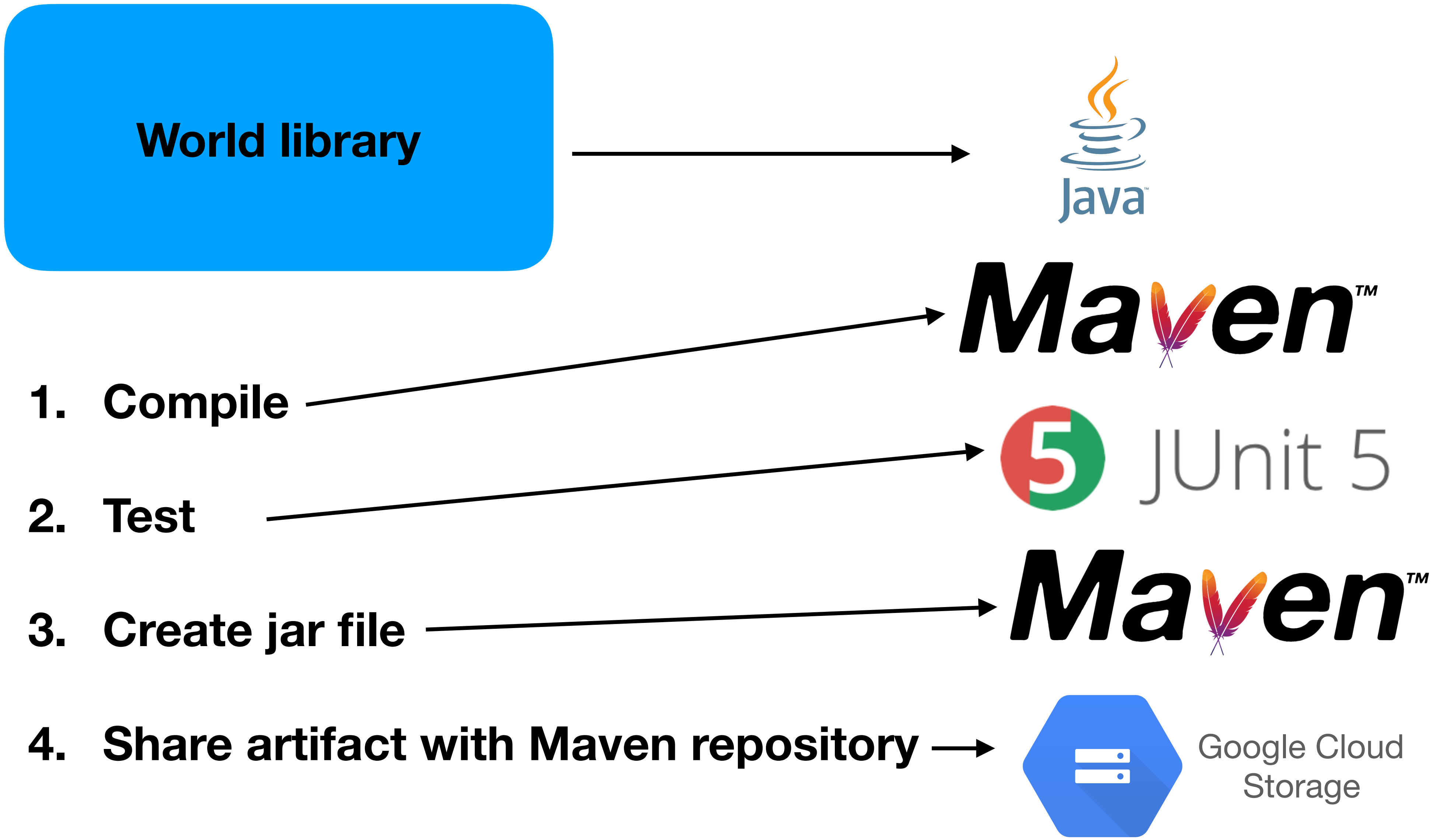
World library

1. Compile
2. Test
3. Create jar file
4. Upload to artifact repository

Hello World service

1. Satisfy dependencies
2. Compile
3. Test
4. Create Java native image
5. Create Docker image
6. Upload Docker image to registry
7. Run

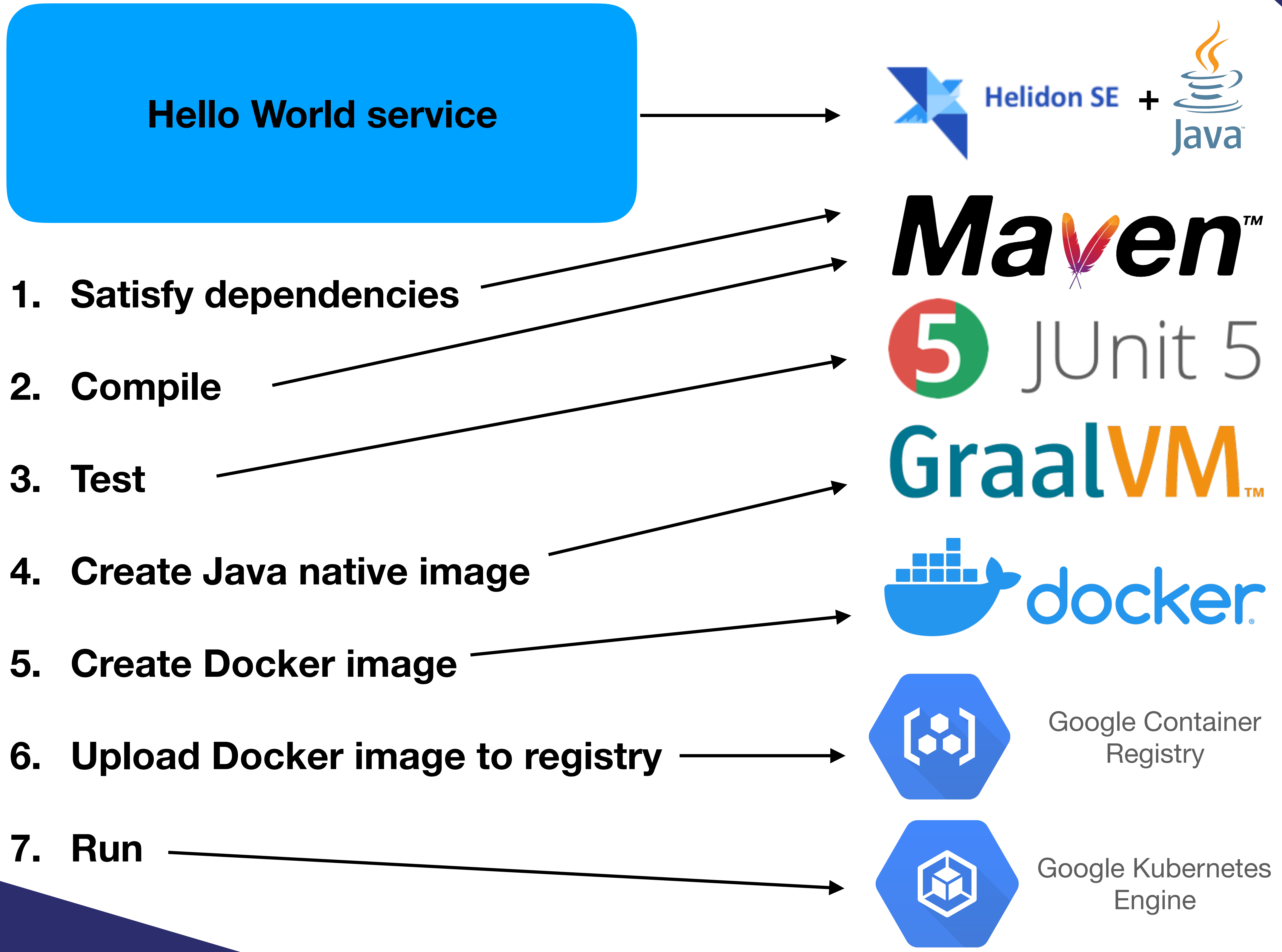
Frameworks and tools



World library → Java

1. **Compile** → Maven
2. **Test** → JUnit 5
3. **Create jar file** → Maven
4. **Share artifact with Maven repository** → Google Cloud Storage

Frameworks and tools



Hello World service

1. Satisfy dependencies
2. Compile
3. Test
4. Create Java native image
5. Create Docker image
6. Upload Docker image to registry
7. Run

Other tools

Dev environment

Visual Studio Code

Cloud IaC provisioning

Terraform

Lab environment

VAGRANT

Continuous Integration

Google Cloud Build

Source repo

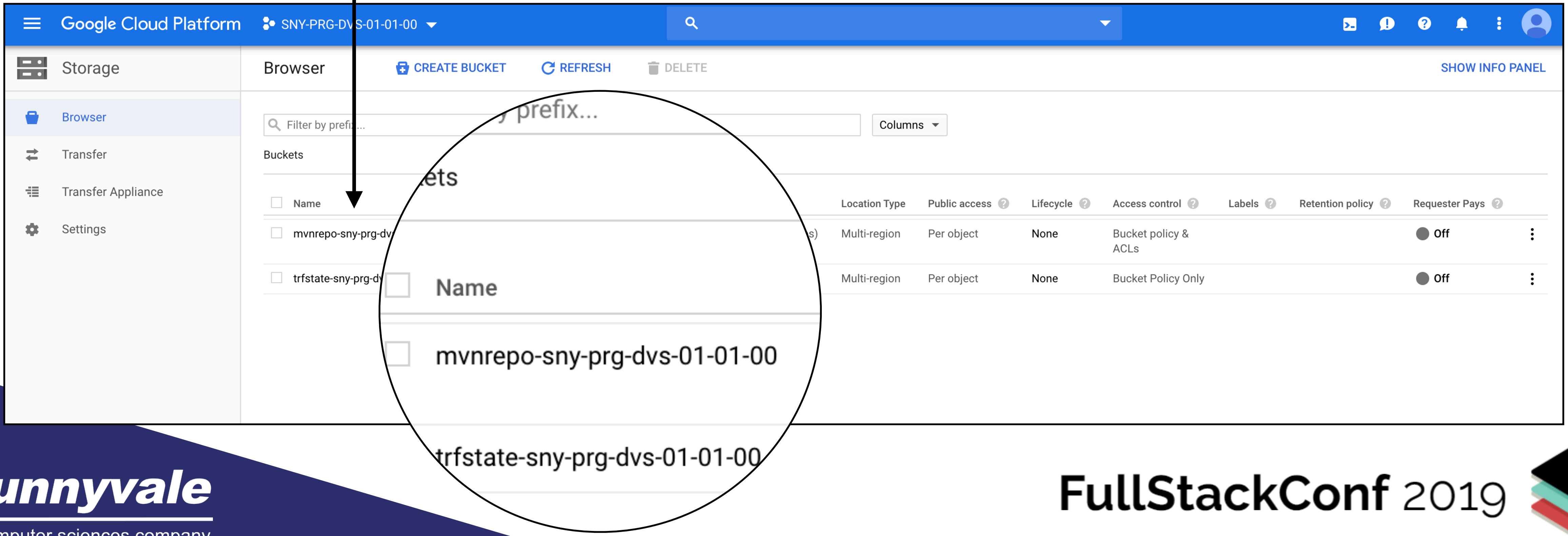
GitHub

Provision Maven artifact repository

```
$ cat maven_repo_bucket.tf
```

```
resource "google_storage_bucket" "mvnrepo" {
  force_destroy = true
  name          = "mvnrepo-${var.PROJECT_ID}"
}
```

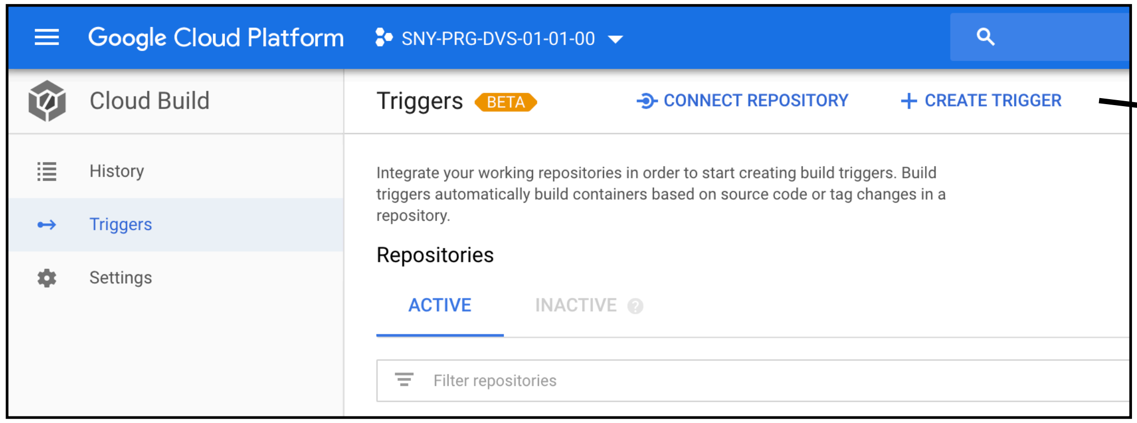
```
$ terraform apply
```



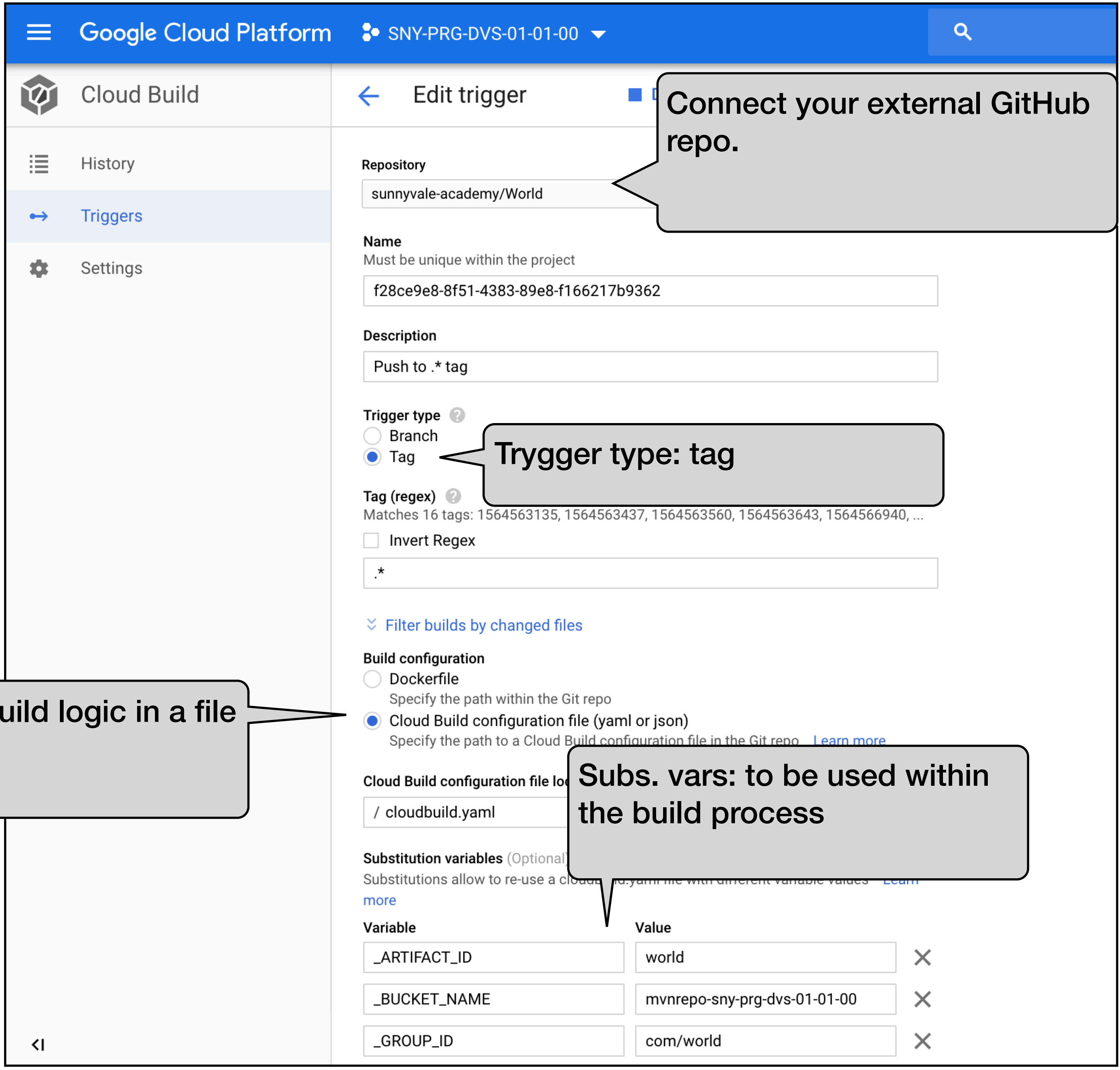
Buckets

Name	Location Type	Public access	Lifecycle	Access control	Labels	Retention policy	Requester Pays
☐ mvnrepo-sny-prg-dvs-01-01-00	Multi-region	Per object	None	Bucket policy & ACLs			● Off
☐ trfstate-sny-prg-dvs-01-01-00	Multi-region	Per object	None	Bucket Policy Only			● Off

The “World” trigger on Cloud Build



Integrate your working repositories in order to start creating build triggers. Build triggers automatically build containers based on source code or tag changes in a repository.



Repository
sunnyvale-academy/World

Name
Must be unique within the project
f28ce9e8-8f51-4383-89e8-f166217b9362

Description
Push to .* tag

Trigger type
○ Branch
● Tag

Tag (regex)
Matches 16 tags: 1564563135, 1564563437, 1564563560, 1564563643, 1564566940, ...
☐ Invert Regex
.*

Build configuration
○ Dockerfile
Specify the path within the Git repo
● Cloud Build configuration file (yaml or json)
Specify the path to a Cloud Build configuration file in the Git repo. Learn more

Cloud Build configuration file lo
/ cloudbuild.yaml

Substitution variables (Optional)
Substitutions allow to re-use a cloud ld.yaml file with different variable values. Learn more

Variable	Value
_ARTIFACT_ID	world
_BUCKET_NAME	mvnrepo-sny-prg-dvs-01-01-00
_GROUP_ID	com/world

Connect your external GitHub repo.

Trygger type: tag

We choose to describe the build logic in a file called cloudbuild.yaml

Subs. vars: to be used within the build process

Sunnyvale
A computer sciences company

FullStackConf 2019

Code the “World” project build steps

```
$ cat World/cloudbuild.yaml
```

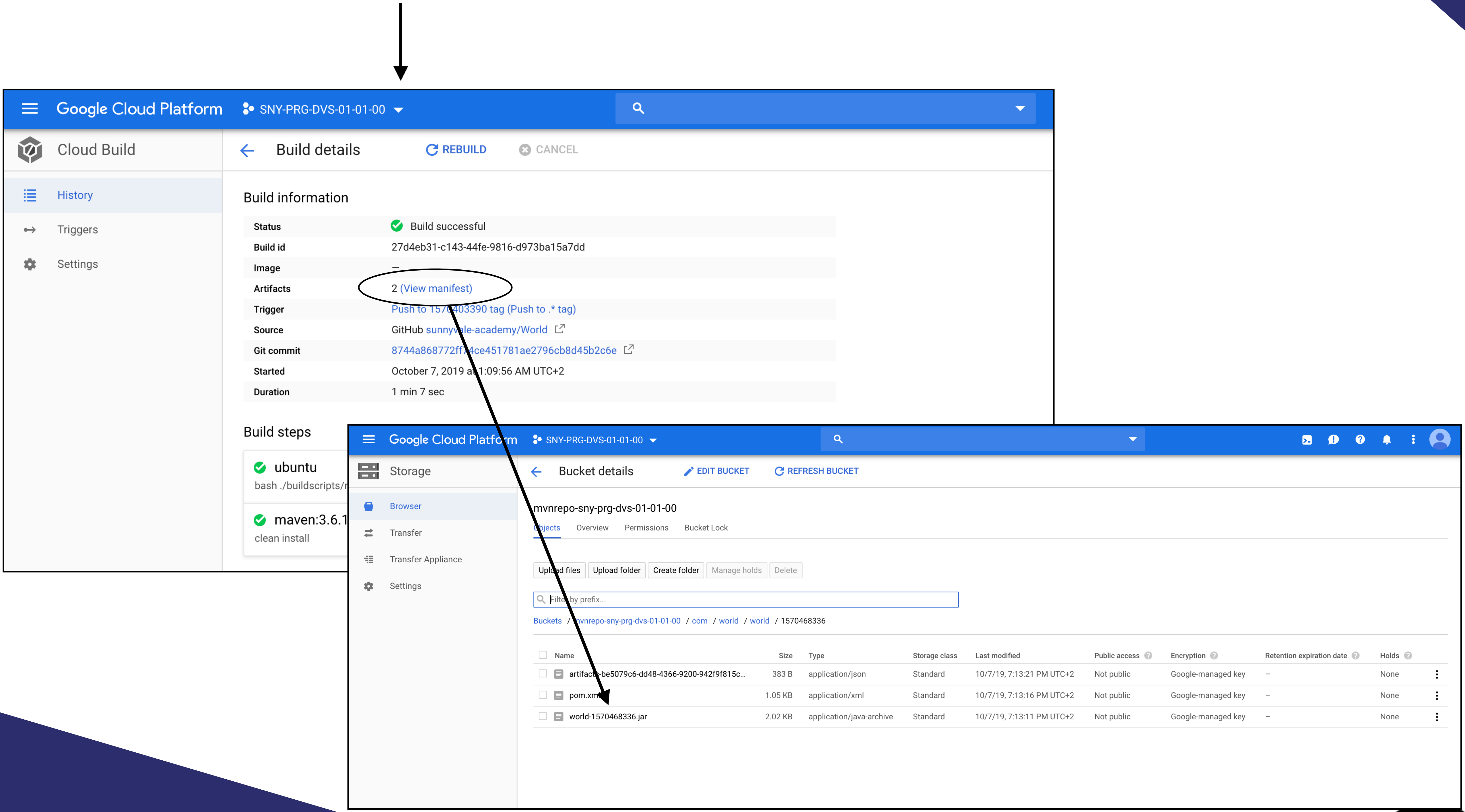
```
steps:
  - name: 'ubuntu'
    args: ['bash', './buildscripts/replace_pom_version.sh']
    env:
     - 'TAG_NAME=$TAG_NAME'
  - name: maven:3.6.1-jdk-12
    entrypoint: 'mvn'
    args: ['clean', 'install']
artifacts:
  objects:
    location: 'gs://${_BUCKET_NAME}/${_GROUP_ID}/${_ARTIFACT_ID}/$TAG_NAME'
    paths: ['target/*.jar','pom.xml']
```

Steps:
- **Insert tag id (version) in pom.xml file**
- **Build the project (`mvn clean install`)**

Artifacts: **Publish jar an pom files to GCS bucket**

“World” project build

```
$ git tag "1570468336" && git push origin --tags
```



Build information

Status	✓ Build successful
Build id	27d4eb31-c143-44fe-9816-d973ba15a7dd
Image	–
Artifacts	2 (View manifest)
Trigger	Push to 1570403390 tag (Push to .* tag)
Source	GitHub sunnyvale-academy/World
Git commit	8744a868772ff74ce451781ae2796cb8d45b2c6e
Started	October 7, 2019 at 1:09:56 AM UTC+2
Duration	1 min 7 sec

Build steps

✓ ubuntu
bash ./buildscripts/

✓ maven:3.6.1
clean install

mvnrepo-sny-prg-dvs-01-01-00

Buckets / mvnrepo-sny-prg-dvs-01-01-00 / com / world / world / 1570468336

☐	Name	Size	Type	Storage class	Last modified	Public access	Encryption	Retention expiration date	Holds
☐	artifacts-be5079c6-dd48-4366-9200-942f9f815c...	383 B	application/json	Standard	10/7/19, 7:13:21 PM UTC+2	Not public	Google-managed key	–	None
☐	pom.xml	1.05 KB	application/xml	Standard	10/7/19, 7:13:16 PM UTC+2	Not public	Google-managed key	–	None
☐	world-1570468336.jar	2.02 KB	application/java-archive	Standard	10/7/19, 7:13:11 PM UTC+2	Not public	Google-managed key	–	None

“Helidon_HelloWorld” pom.xml

```
$ cat Helidon_HelloWorld/pom.xml
```

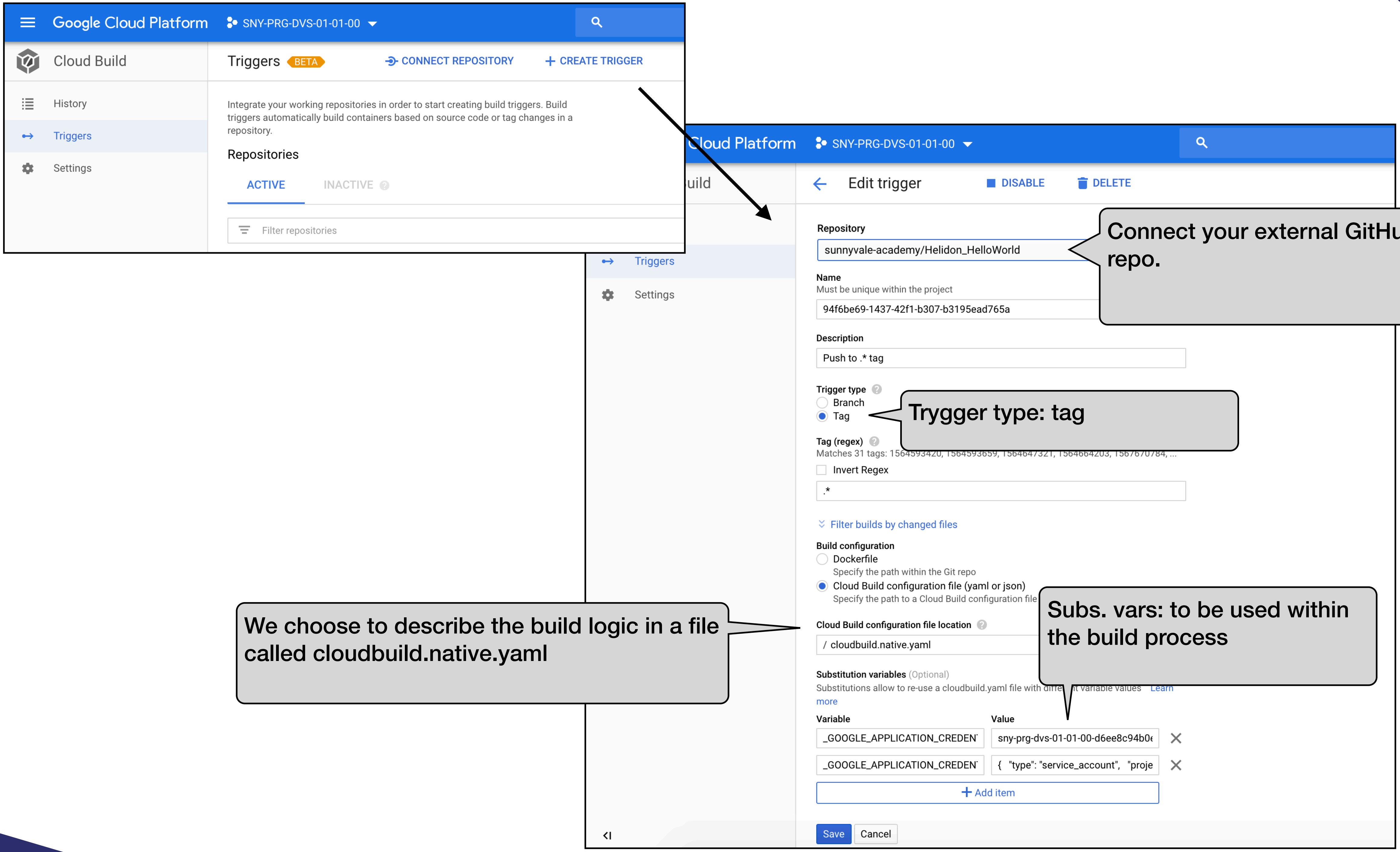
```
…
    <extensions>
          <extension>
              <groupId>com.gkatzioura.maven.cloud</groupId>
              <artifactId>google-storage-wagon</artifactId>
              <version>1.0</version>
          </extension>
    </extensions>
…
    <repositories>
        <repository>
            <id>my-repo-bucket-snapshot</id>
            <url>gs://mvnrepo-sny-prg-dvs-01-01-00</url>
        </repository>
        <repository>
            <id>my-repo-bucket-release</id>
            <url>gs://mvnrepo-sny-prg-dvs-01-01-00</url>
        </repository>
    </repositories>
…
    <dependencies>
        <dependency>
            <groupId>com.world</groupId>
            <artifactId>world</artifactId>
            <version>1570468336</version>
        </dependency>
    </dependencies>
```

Declare the Maven extension

Point to GCS bucket

Declare the World dep. release: 1570468336

HelloWorld” trigger on Cloud Build



Integrate your working repositories in order to start creating build triggers. Build triggers automatically build containers based on source code or tag changes in a repository.

Repositories

Repository
sunnyvale-academy/Helidon_HelloWorld

Name
Must be unique within the project
94f6be69-1437-42f1-b307-b3195ead765a

Description
Push to .* tag

Trigger type
○ Branch
● Tag

Tag (regex)
Matches 31 tags: 1564593420, 1564593659, 1564647321, 1564664203, 1567670784, ...
☐ Invert Regex
.*

Filter builds by changed files

Build configuration
○ Dockerfile
Specify the path within the Git repo
● Cloud Build configuration file (yaml or json)
Specify the path to a Cloud Build configuration file

Cloud Build configuration file location
/ cloudbuild.native.yaml

Substitution variables (Optional)
Substitutions allow to re-use a cloudbuild.yaml file with different variable values Learn more

Variable	Value
_GOOGLE_APPLICATION_CREDEN	sny-prg-dvs-01-01-00-d6ee8c94b0
_GOOGLE_APPLICATION_CREDEN	{ "type": "service_account", "proje

Connect your external GitHub repo.

Trygger type: tag

We choose to describe the build logic in a file called cloudbuild.native.yaml

Subs. vars: to be used within the build process

The “Helidon_HelloWorld” build steps

```
$ cat Helidon_HelloWorld/cloudbuild.native.yaml
```

```
steps:
  - name: 'ubuntu'
    args: ['bash', './buildscripts/create_json_auth_file.sh']
    env:
      - 'GOOGLE_APPLICATION_CREDENTIALS_CONTENT=$
{_GOOGLE_APPLICATION_CREDENTIALS_CONTENT}'
      - 'GOOGLE_APPLICATION_CREDENTIALS=${_GOOGLE_APPLICATION_CREDENTIALS}'
  - name: 'gcr.io/cloud-builders/docker'
    args: ['build', '-f', 'Dockerfile.native', '--build-arg',
'GOOGLE_APPLICATION_CREDENTIALS=${_GOOGLE_APPLICATION_CREDENTIALS}', '-t',
'gcr.io/sny-prg-dvs-01-01-00/helidon_helloworld:$TAG_NAME', '.']
    env:
      - 'GOOGLE_APPLICATION_CREDENTIALS=${_GOOGLE_APPLICATION_CREDENTIALS}'
  - name: 'gcr.io/cloud-builders/docker'
    args: ['push', 'gcr.io/sny-prg-dvs-01-01-00/helidon_helloworld:$TAG_NAME']
```

Steps:

- **Read the substitution var and create service account json file**
- **Build the project, create the Java native image (with GraalVM), build the docker image**
- **Push Docker image to registry**

“Helidon_HelloWorld” Dockerfile.native*

```
$ cat Helidon_HelloWorld/Dockerfile.native
```

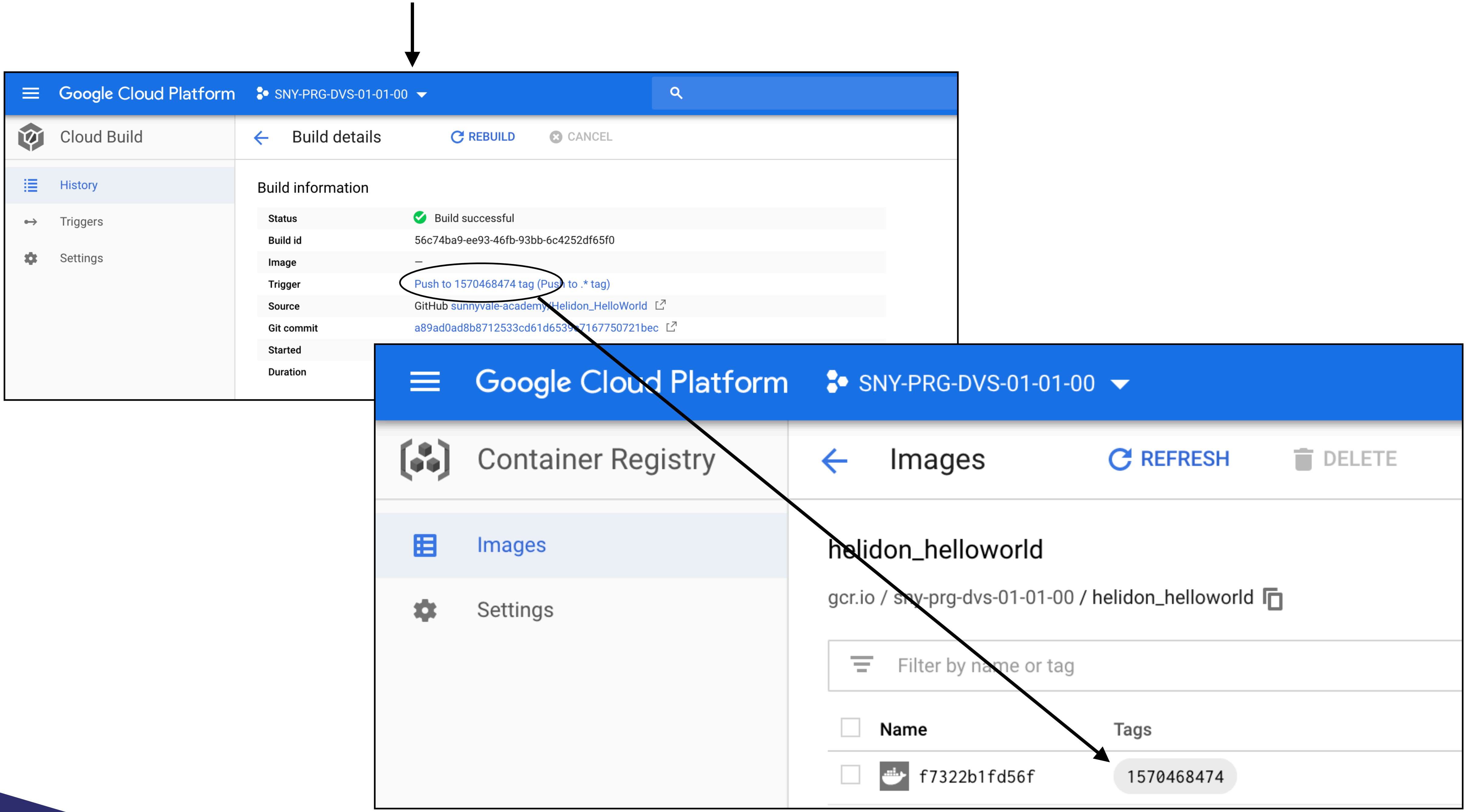
```
FROM helidon/jdk8-graalvm-maven:19.2.0 as build
RUN mvn package -Pnative-image -Dnative.image.buildStatic -DskipTests

FROM scratch
COPY --from=build /workspace/target/Helidon_HelloWorld .
ENTRYPOINT ["./Helidon_HelloWorld"]
EXPOSE 8080
```

***File Dockerfile.native has been shortened in size for the sake of clarity**

“Helidon_HelloWorld” project build

```
$ git tag "1570468474" && git push origin --tags
```



Build information

Status	✓ Build successful
Build id	56c74ba9-ee93-46fb-93bb-6c4252df65f0
Image	–
Trigger	Push to 1570468474 tag (Push to .* tag)
Source	GitHub sunnyvale-academy/Helidon_HelloWorld
Git commit	a89ad0ad8b8712533cd61d65397167750721bec
Started	
Duration	

helidon_helloworld

gcr.io / sny-prg-dvs-01-01-00 / helidon_helloworld

Filter by name or tag

☐ Name	Tags
☐ f7322b1fd56f	1570468474

Sunnyvale
A computer sciences company

FullStackConf 2019

Run a “Helidon_HelloWorld” container

```
$ docker run -ti -p 8080:8080 gcr.io/sny-prg-dvs-01-01-00/
helidon_helloworld:1570468474
```

```
…
WEB server is up! http://localhost:8080/greet
```

gcr.io = Google Container Registry base URL
sny-prg-dvs-01-01-00 = GCP project id
helidon_helloworld = image name
1570468474 = image release

```
$ curl localhost:8080/greet
```

```
{"message":"Hello  world!!"}
```

Native vs non-native image size

Native compiling using cloudbuild.native.yaml (GraalVM)

```
$ docker images | grep “helidon_helloworld:1570468474”
```

```
REPOSITORY                                      TAG                        SIZE
gcr.io/sny-prg-dvs-01-01-00/helidon_helloworld   1570468474                 21.3MB
```

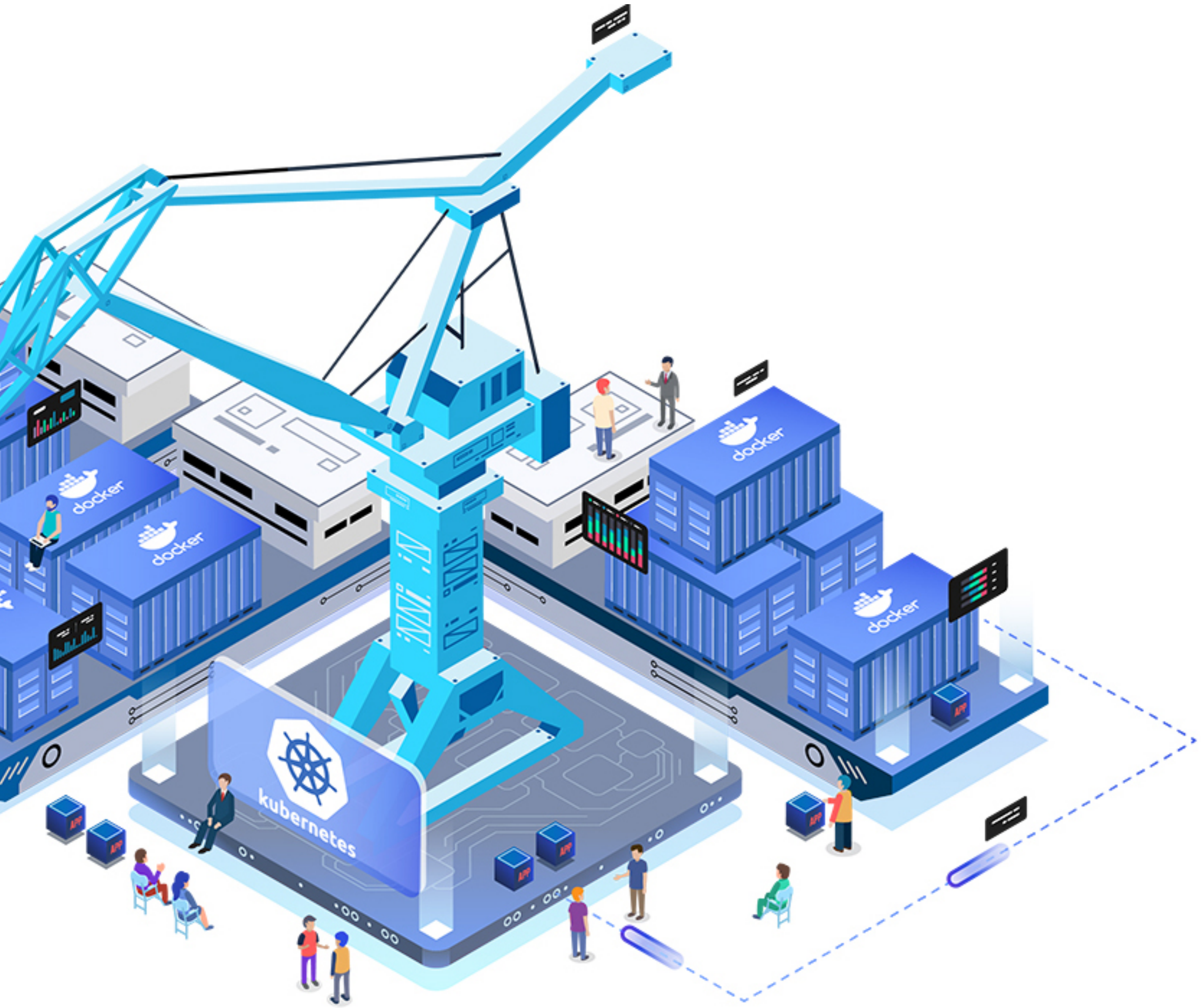
Non-native compiling cloudbuild.yaml (OpenJDK)

```
$ docker images | grep “helidon_helloworld:1564593659”
```

```
REPOSITORY                                      TAG                        SIZE
gcr.io/sny-prg-dvs-01-01-00/helidon_helloworld   1570468474                 476MB
```

And now?

Run our microservice on Kubernetes



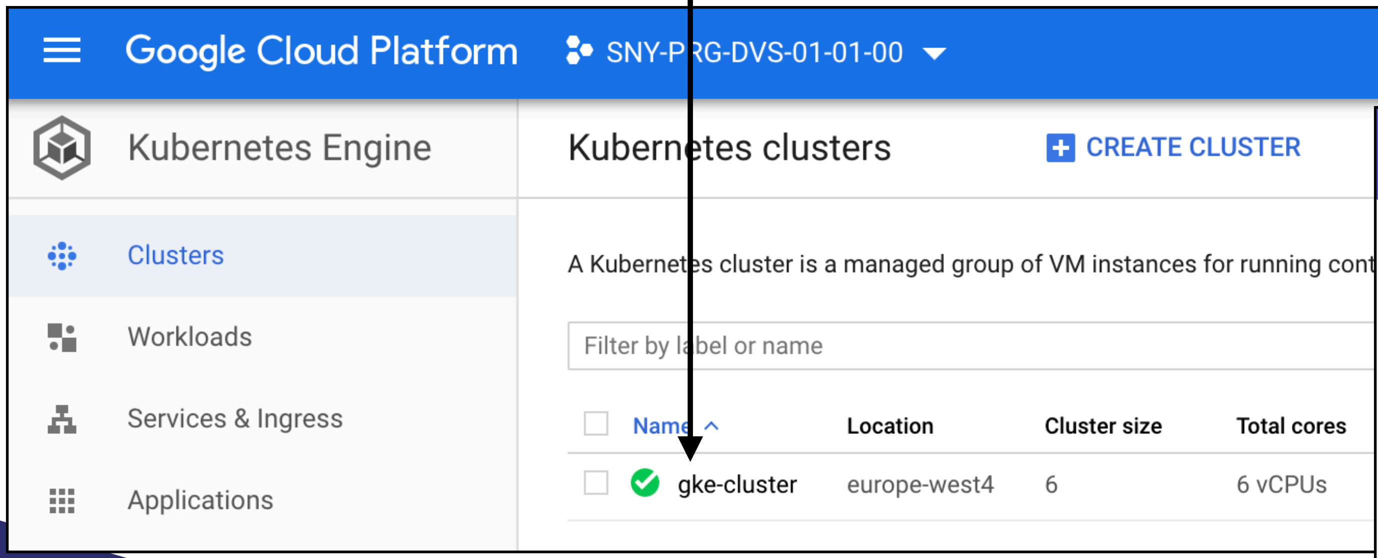
Google Kubernetes Engine

Provision GKE cluster

```
$ cat gke_cluster.tf
module "gke" {
  source   = "terraform-google-modules/kubernetes-engine/google"
  version  = "5.0.0"
  name     = "gke-cluster"
}
```

```
$ terraform apply
```

```
$ terraform init
```

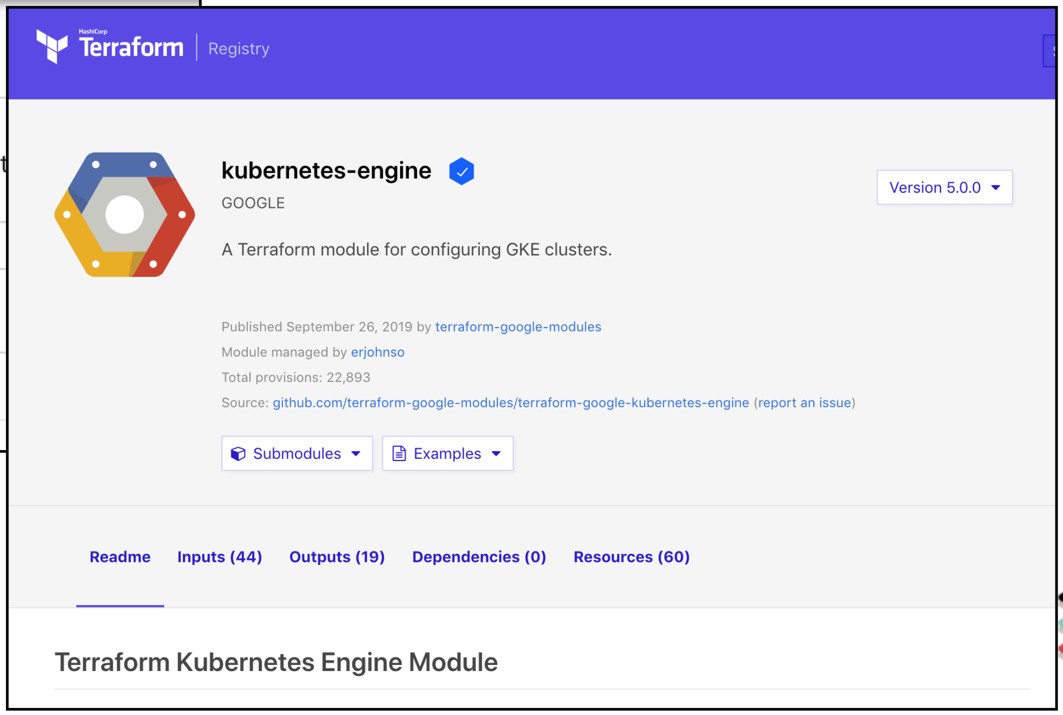


Kubernetes clusters

A Kubernetes cluster is a managed group of VM instances for running cont

Filter by label or name

☐	Name	Location	Cluster size	Total cores
☐	✓ gke-cluster	europe-west4	6	6 vCPUs



kubernetes-engine

GOOGLE

A Terraform module for configuring GKE clusters.

Published September 26, 2019 by terraform-google-modules
Module managed by erjohnso
Total provisions: 22,893
Source: github.com/terraform-google-modules/terraform-google-kubernetes-engine (report an issue)

Terraform Kubernetes Engine Module

Create a Deployment

```
$ cat deployment.yaml
```

```
kind: Deployment
…
spec:
  replicas: 5
  …
      containers:
      - image: gcr.io/sny-prg-dvs-01-01-00/helidon_helloworld:1570468474
        imagePullPolicy: IfNotPresent
        name: web
        ports:
        - containerPort: 8080
          protocol: TCP
```

```
$ kubectl apply -f deployment.yaml
```

```
deployment.extensions/web created
```

```
$ kubectl get pods
```

```
NAME                       READY   STATUS    RESTARTS   AGE
web-7c8f7f7c44-h5pj2       1/1     Running   0          24s
web-7c8f7f7c44-hm9ps       1/1     Running   0          24s
web-7c8f7f7c44-n97mn       1/1     Running   0          24s
web-7c8f7f7c44-p2kms       1/1     Running   0          24s
web-7c8f7f7c44-wv2v7       1/1     Running   0          24s
```

Create the NodePort Service

```
$ cat service.yaml
```

```
apiVersion: v1
kind: Service
…
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    run: web
  type: NodePort
```

```
$ kubectl apply -f service.yaml
```

```
service/web created
```

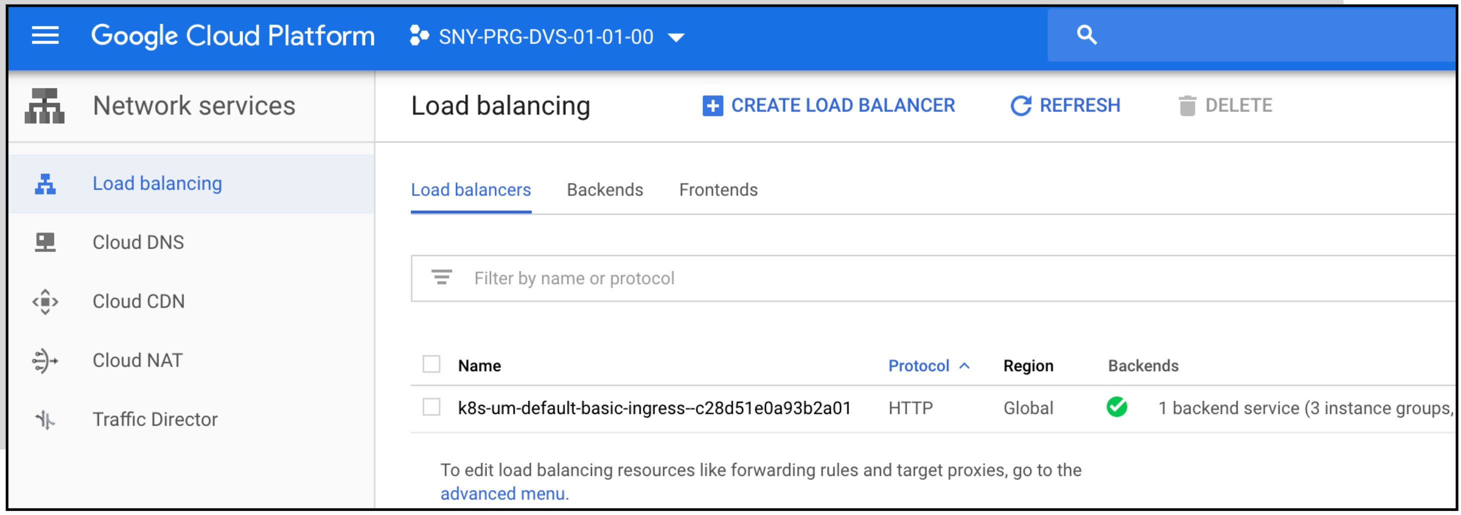
```
$ kubectl get services
```

```
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP    PORT(S)
web          NodePort    10.78.13.45    <none>         8080:32489/TCP
```

Create the Ingress (and LBaaS)

```
$ cat ingress.yaml
```

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: basic-ingress
spec:
  backend:
    serviceName: web
    servicePort: 8080
```



☐	Name	Protocol ^	Region	Backends
☐	k8s-um-default-basic-ingress--c28d51e0a93b2a01	HTTP	Global	✓ 1 backend service (3 instance groups,

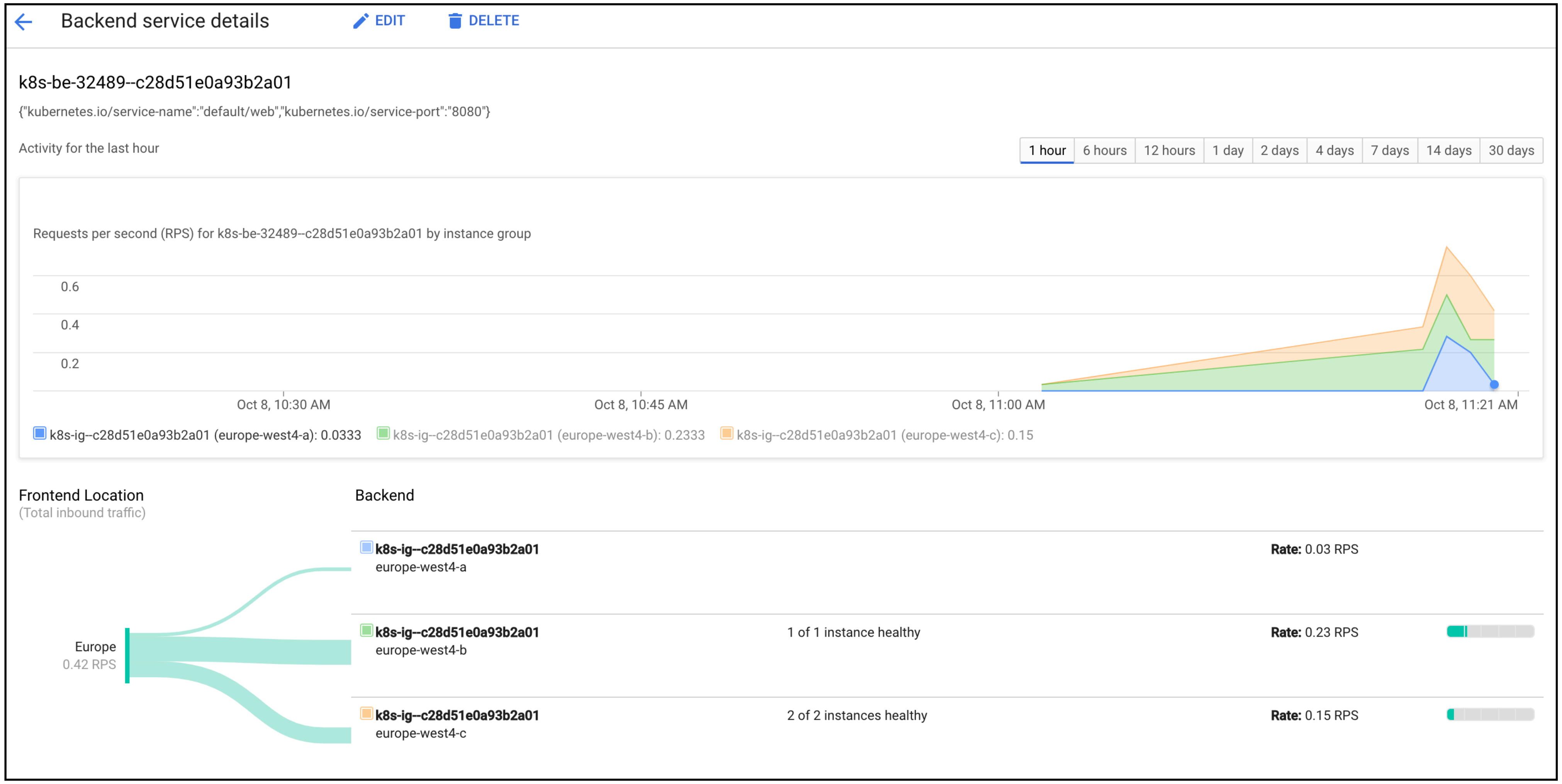
To edit load balancing resources like forwarding rules and target proxies, go to the advanced menu.

```
$ kubectl apply -f ingress.yaml
ingress.extensions/basic-ingress created
```

```
$ kubectl get ingresses
NAME           HOSTS    ADDRESS        PORTS    AGE
basic-ingress  *        35.241.55.77   80       119s
```

Test “Helidon_HelloWorld” on GKE

```
$ while true; do curl http://35.241.55.77/greet; done
{"message":"Hello  world!!"}{"message":"Hello  world!!”}…
```



Backend service details

k8s-be-32489--c28d51e0a93b2a01

{"kubernetes.io/service-name":"default/web","kubernetes.io/service-port":"8080"}

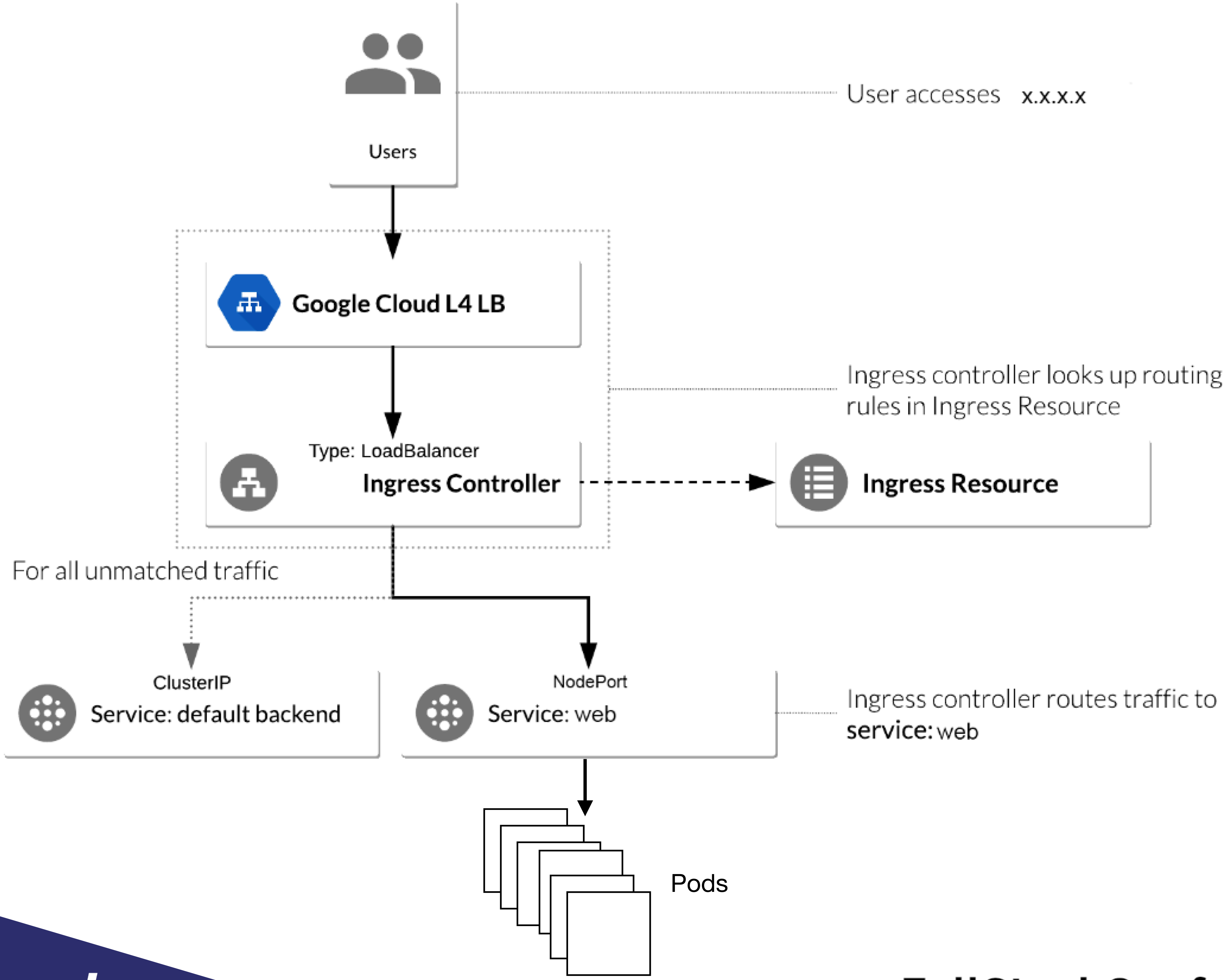
Activity for the last hour

Requests per second (RPS) for k8s-be-32489--c28d51e0a93b2a01 by instance group

🟦 k8s-ig--c28d51e0a93b2a01 (europe-west4-a): 0.0333
🟩 k8s-ig--c28d51e0a93b2a01 (europe-west4-b): 0.2333
🟧 k8s-ig--c28d51e0a93b2a01 (europe-west4-c): 0.15

Frontend Location (Total inbound traffic)	Backend		
Europe 0.42 RPS	**k8s-ig--c28d51e0a93b2a01** europe-west4-a		**Rate:** 0.03 RPS
	k8s-ig--c28d51e0a93b2a01 europe-west4-b	1 of 1 instance healthy	**Rate:** 0.23 RPS
	k8s-ig--c28d51e0a93b2a01 europe-west4-c	2 of 2 instances healthy	**Rate:** 0.15 RPS

Final microservice architecture



Users

User accesses x.x.x.x

Google Cloud L4 LB

Ingress controller looks up routing rules in Ingress Resource

Type: LoadBalancer

Ingress Controller

Ingress Resource

For all unmatched traffic

ClusterIP

Service: default backend

NodePort

Service: web

Ingress controller routes traffic to service: web

Pods

"Talk is cheap, show Me the code"

- Linus Torvalds

Source code is available on GitHub

https://github.com/sunnyvale-academy/SNY.PRG.DVS.01.01.00
https://github.com/sunnyvale-academy/Helidon_HelloWorld
https://github.com/sunnyvale-academy/World

Fork me on GitHub

Thanks!

Denis Maggiorotto

denis.maggiorotto@sunnyvale.it

twitter.com/denismaggior8

www.linkedin.com/in/denismaggiorotto

www.github.com/denismaggior8