Anthos Service Mesh
Google confidential & proprietary

All content is for training purposes only

Please DO NOT

- Post on Social Media
- Share unless granted permission by Google
How to build **globally scalable distributed services** on GCP
Principles

❖ Opinionated
❖ Managed open source
❖ Not an Istio feature list
❖ Production “worthy”
  ■ Secure
  ■ Scalable
  ■ Tooling
  ■ Declarative
  ■ Automation
Agenda

● **Module 0 - Introduction and Platform Setup**
  ○ Intro and architecture
  ○ Intro to Service Mesh
  ○ **Lab: Infrastructure Setup: User workflow**
    Break
  ○ Intro to Istio and Anthos Service Mesh (ASM)

● **Module 1 - Install, secure and monitor applications with ASM**
  ○ 2 Repo model: Infrastructure and Kubernetes repos explained - the GitOps way
  ○ **Lab: Deploy sample application**
  ○ Distributed Services and Observability
    Lunch
  ○ **Lab: Observability with Stackdriver**

● **Module 2 - DevOps - Canary rollouts, policy/RBAC**
  ○ Multi cluster service discovery and policy (AuthZ and AuthN)
  ○ **Lab: Mutual TLS**
  ○ Canary Deployments with Istio
  ○ **Lab: Canary Deployments**
  ○ Secure multi cluster global load balancing
    Break
  ○ **Lab: Authorization Policy**

● **Module 3 - Infra Ops - Platform upgrades**
  ○ Distributed Service building blocks
  ○ **Lab: Infrastructure Scaling**
  ○ Whats next?
  ○ Feedback and Prizes!!
  ○ Happy hour

● **Future Module 4 - Infra Ops continued - Platform resiliency**
Personas and user journeys
Flexible

Compute Engine 2012

Kubernetes Engine 2015

App Engine 2008

Opinionated
Your platform built on Kubernetes

- Compute Engine 2012
- Kubernetes Engine 2015
- App Engine 2008

Flexible - Opinionated
Infrastructure

projects, VPCs, compute, storage, clusters

Apps
Infrastructure

projects, VPCs, compute, storage, clusters

Platform

deploy tools, service mgt, policy

Apps
projects, VPCs, compute, storage, clusters

deploy tools, service mgt, policy

infra

platform / SRE

dev

.tf

.yaml

.go

.java

Apps

Platform

Infrastructure
Build → Monitor → Govern → Scale → Upgrade

Own GCP resources - projects, VPC, clusters

Operate & troubleshoot & repair & recover
Own Kubernetes resources - CI/CD tools, service mesh, applications

Deploy / Migrate → Monitor → Govern → Connect → Rollout

Operate & troubleshoot & repair & recover

Own GCP resources - projects, VPC, clusters

Build → Monitor → Govern → Scale → Upgrade

Operate & troubleshoot & repair & recover
Own **Service resources** - containers, configurations, biz logic

1. Code → Test → Build → Release → Debug

2. Operate & troubleshoot & repair & recover

Own **Kubernetes resources** - CI/CD tools, service mesh, applications

3. Deploy / Migrate → Monitor → Govern → Connect → Rollout

4. Operate & troubleshoot & repair & recover

Own **GCP resources** - projects, VPC, clusters

5. Build → Monitor → Govern → Scale → Upgrade

6. Operate & troubleshoot & repair & recover
shared-vpc
shared-vpc

project-2-dev1-apps

subnet-3 with secondary ranges in region 1

project-3-dev2-apps

subnet-4 with secondary ranges in region 2

project-1-ops-asm

subnet-1 with secondary ranges in region 1

subnet-2 with secondary ranges in region 2
shared-vpc

project-2-dev1-apps

subnet-3 with secondary ranges in region 1

- gke-2-apps-r1b-prod (region-1-zone-b, apps)
- gke-1-apps-r1a-prod (region-1-zone-a, apps)

project-3-dev2-apps

subnet-4 with secondary ranges in region 2

- gke-4-apps-r2b-prod (region-2-zone-b, apps)
- gke-3-apps-r2a-prod (region-2-zone-c, apps)

project-1-ops-asm

subnet-1 with secondary ranges in region 1

- gke-asm-1-r1-prod (region-1-regional, controlplane)

subnet-2 with secondary ranges in region 2

- gke-asm-2-r2-prod (region-2-regional, controlplane)
project-2-dev1-apps

subnet-3 with secondary ranges in region 1

gke-2-apps-r1b-prod (region-1-zone-b, apps)
gke-1-apps-r1a-prod (region-1-zone-a, apps)

project-3-dev2-apps

subnet-4 with secondary ranges in region 2

gke-4-apps-r2b-prod (region-2-zone-b, apps)
gke-3-apps-r2a-prod (region-2-zone-c, apps)

project-1-ops-asm

subnet-1 with secondary ranges in region 1

gke-asn-1-r1-prod (region-1-regional, controlplane)

istio install (multicluster + mesh expansion, mTLS)

subnet-2 with secondary ranges in region 2

gke-asn-2-r2-prod (region-2-regional, controlplane)

istio install (multicluster + mesh expansion, mTLS)
shared-vpc

**project-2-dev1-apps**

- subnet-3 with secondary ranges in region 1
  - gke-2-apps-r1b-prod (region-1-zone-b, apps)
  - gke-1-apps-r1a-prod (region-1-zone-a, apps)

**mesh-region-1**
(shared controlplane)

**project-3-dev2-apps**

- subnet-4 with secondary ranges in region 2
  - gke-4-apps-r2b-prod (region-2-zone-b, apps)
  - gke-3-apps-r2a-prod (region-2-zone-c, apps)

**mesh-region-2**
(shared controlplane)

**project-1-ops-asm**

- subnet-1 with secondary ranges in region 1
  - gke-asm-1-r1-prod (region-1-regional, controlplane)

**istio install** (multicluster + mesh expansion, mTLS)

**project-2-dev1-apps**

- subnet-2 with secondary ranges in region 2
  - gke-asm-2-r2-prod (region-2-regional, controlplane)

**istio install** (multicluster + mesh expansion, mTLS)
shared-vpc

project-2-dev1-apps

subnet-3 with secondary ranges in region 1

mesh-region-1
(shared controlplane)

- gke-2-apps-r1b-prod (region-1-zone-b, apps)
  Hipster Shop

- gke-1-apps-r1a-prod (region-1-zone-a, apps)
  Hipster Shop

project-3-dev2-apps

subnet-4 with secondary ranges in region 2

mesh-region-2
(shared controlplane)

- gke-4-apps-r2b-prod (region-2-zone-b, apps)
  Hipster Shop

- gke-3-apps-r2a-prod (region-2-zone-c, apps)
  Hipster Shop

project-1-ops-asm

subnet-1 with secondary ranges in region 1

- gke-asm-1-r1-prod (region-1-regional, controlplane)
  istio install (multicluster + mesh expansion, mTLS)

subnet-2 with secondary ranges in region 2

- gke-asm-2-r2-prod (region-2-regional, controlplane)
  istio install (multicluster + mesh expansion, mTLS)
Architecture best practices

1. Use **shared VPC**
2. Separate **projects** by function/team (billing, IAM, quota)
3. Separate **environments** (dev, stage, prod) into separate projects
4. Use **separate projects and clusters** for ops and applications
5. Use **multiple clusters in multiple zones** in a single region for applications (zonal redundancy)
6. Use **multiple clusters in multiple regions** for applications (regional redundancy)
7. Use **VPC native clusters** (Pod to Pod and Pod to VM communication across clusters)
8. Use a separate **project to provision** and manage all GCP resources (terraform-admin-project)
9. Consider **failure domains and redundancy** at every layer (regional, zonal, cluster, control planes)
10. **Infrastructure should scale easily** (in any dimension)
11. **Carefully consider infra boundaries**
Architecture best practices

network

project

cluster

namespace
network

project

cluster

namespace

frontend

jenkins

project

cluster

namespace
Intro to service mesh
We are operators!
Monolith

function(a)  function(b)
container

function(a)

container

function(b)
Network

container A

container B
Zero trust network
Container A

Security

Network Resilience

Policy

API

Container B

Security

Network Resilience

Policy

API

Scripts:
- Authn
- Authz
- Latency
- Fault Tolerance
- Circuit Breaking
- Quota
- Rate Limiting

ID

Observability

ID

Observability
Not business logic

Security

Business logic

API

Network Resilience

Policy

Authn
Authz
Latency
Fault Tolerance
Circuit Breaking
Quota
Rate Limiting
Logging
Metrics
Distributed Tracing
Topology

ID

API

ID

Security

Business logic

Network Resilience

Policy

Observability

Observability
Network Functions

- Security
- Business logic
- API

Observability
- Network Resilience
- Policy

Security
- Authn
- Authz
- Latency
- Fault Tolerance
- Circuit Breaking
- Quota
- Rate Limiting
- Logging
- Metrics
- Distributed Tracing
- Topology

Network Resilience

Policy

Observability
Service Mesh

Separating applications from network functions
Business logic

Security

Network Resilience

Policy

Observability
Pod

- Container A
- Envoy (sidecar)
- App

Lightweight
Portable

- Observability
- Security
- Network Resilience
- Policy

Dev
Devops
Pod

container A

app

Envoy (sidecar)

Security

Network Resilience

Policy

Observability

Other” functionality

Lightweight

Portable

Automation

Scalability

Consistency

Control
Lab 4/5: Prep & Infrastructure Setup: User workflow

Objective: Workshop prep and verify infrastructure and Istio installation

- Fast Track vs Copy-and-paste Lab methods
- Install workshop tools
- Clone workshop repo
- Verify Infrastructure install
- Verify k8s-repo install
- Verify Istio installation

20 mins
ASM controlplane &
life of a request
Svc A

app

envoy

?

Svc B

app

envoy

data plane

control plane
Svc A

app

envoy

Svc B

app

envoy

pilot

data plane

control plane
Service Discovery
k8s, consul, zk, custom

Svc A
app
envoy

Svc B
app
envoy
Service Discovery
k8s, consul, zk, custom

Svc A
app
envoy

Svc B
app
envoy

control plane
data plane
Service Discovery
k8s, consul, zk, custom

Svc A
  app
  envoy

Svc B
version: prod
version: canary

pilot

control plane

data plane
Service Discovery
k8s, consul, zk, custom

Svc A
  app
  envoy

Svc B
  version: prod
  version: canary

Traffic Rules

pilot

data plane

control plane
Service Discovery
k8s, consul, zk, custom

Svc A
app
envoy

Svc B
version: prod

Svc B
version: canary

Traffic Rules
90%
10%
Service Discovery
k8s, consul, zk, custom

Traffic Rules

Svc A

app
tenvoy

Svc B

version: prod

version: canary

pilot

control plane
data plane

inject 10ms delay

90% 10%
Service Discovery
k8s, consul, zk, custom

Traffic Rules

Svc A
app
svc

Svc B
version: prod

Svc B
version: canary

90%

10% HTTP 500s
inject 10ms delay
10%
Service Discovery
k8s, consul, zk, custom

Svc A

app

Svc B

version: prod

90%

Svc B

version: canary

10%

Svc B

version: dev

Traffic Rules

pilot

data plane

control plane
Service Discovery
k8s, consul, zk, custom

Traffic Rules

Svc A

app

envoy

Svc B

Circuit breaker
if 5 consecutive HTTP errors

pilot
Service Discovery
k8s, consul, zk, custom

Traffic Rules

Svc A
- app
- envoy

Svc B

Rate Limit
50 rps

control plane
data plane
Service Discovery
k8s, consul, zk, custom

Traffic Rules

Svc A
app
envoy

Svc B

Svc C
Rate Limit
50 rps

Svc D
Service Discovery
k8s, consul, zk, custom

Svc A
- app
- envoy

Svc B

Svc C

Svc D

Traffic Rules

pilot

mixer

data plane

control plane

Rate Limit
50 rps
Service Discovery
k8s, consul, zk, custom

Svc A
- app
- envoy

Svc B

Svc C

Svc D

Traffic Rules

pilot

mixer

metrics

control plane
data plane

rate limit 50 rps
Service Discovery
k8s, consul, zk, custom

Traffic Rules

Svc A

app

envoy

Svc B

Svc C

Svc D

Mixing

Rate Limit 50 rps

Traffic Rules

Stackdriver / Prometheus / Custom
Service Discovery
k8s, consul, zk, custom

Svc A

app

envoy

Svc B

Svc C

Svc D

Traffic Rules

Stackdriver / Prometheus / Custom

pilot

mixer

control plane
data plane

Svc A not allowed to Svc C

Rate Limit
50 rps
Service Discovery
k8s, consul, zk, custom

Svc A
app
envoy
mixer fat client

Svc B
mixer fat client

Svc C
mixer fat client

Svc D
mixer fat client

Traffic Rules
Stackdriver / Prometheus / Custom

pilot
mixer

control plane
data plane

Svc A not allowed to Svc C
Rate Limit 50 rps
Service Discovery
k8s, consul, zk, custom

Svc A
app
envoy

Traffic Rules

Svc B

Svc C

Svc D

Stackdriver / Prometheus / Custom

mixer

control plane
data plane
Service Discovery
k8s, consul, zk, custom

Svc A

app
envoy

Svc B

Svc C

Svc D

Traffic Rules
Stackdriver / Prometheus / Custom

pilot

mixer

control plane
data plane
Service Discovery
k8s, consul, zk, custom

Svc A
app
envoy

Traffic Rules
pilot

Stackdriver / Prometheus / Custom
mixer
citadel

Svc B
Svc C
Svc D

Traffic Rules
data plane
control plane
Infrastructure and GitOps
CLI / Web Console

imperative scripts
(bash, python)

declarative tools
(deploy mgr, terraform)
CLI / Web Console

imperative scripts (bash, python)

declarative tools (deploy mgr, terraform)

TF files

GKE
CLI / Web Console

imperative scripts
(bash, python)

declarative tools
(deploy mgr, terraform)

TF files

Git

GKE
TF files -> git
TF files → git → builder (container with terraform for example)
TF files -> git (container with terraform for example) -> GKE
multi user and teams

TF files -> git

TF files -> git

TF files -> git

builder (container with terraform for example) -> GKE
multi user and teams

configuration files

TF files

TF files

TF files

git

builder
(container with terraform for example)

GKE
multi user and teams

configuration files

environment

TF files

TF files

TF files

git

builder (container with terraform for example)

GKE
GitOps best practices and benefits

1. **State outside of the environment**
2. **Multi user/team workflow**
3. **Code checks and reviews**
4. **Branches** for testing or environments
5. **Infrastructure-as-Code**
6. **Consistent deployments**
7. **Idempotent**
8. **Audit** (commit logs) and history (rollbacks)
9. **DR / Recovery**
10. **Consistent process** for you teams - Infrastructure, Platforms and Applications
tf-admin (infra/platform)

- infrastructure
- CSR Repo
- Cloud Build
- Cloud Build SA
- GCS
- TF States, logs, vars

OPS (devops)

- gke-ops-1
  - Hipster
- gke-ops-2
  - ASM/Istio
- Cloud Build
- k8s-repo CSR Repo

HOST (network)

- shared VPC

dev1 (team1)

- gke-1
  - Hipster

dev2 (team2)

- gke-3
  - Hipster
- gke-4
  - Hipster

OPC Repo(s)

- dev CSR Repo(s)
Infrastructure deployment best practices and lessons learned

1. Use **Infrastructure as Code** (IaC) and Gitops methodologies (audit/history of changes)
2. **Automate** as much as possible (lower risk of human error)
3. Use **builder containers** (consistently deploy)
4. Use **GCP Service Accounts** to provision resources (IAM)
5. Use **declarative tools** (Terraform, Deployment Manager)
6. Save **state** to GCS buckets (easily share states)
7. Use **folder hierarchy** to separate IaC by team, environment and resource (easily scale)
8. Don’t depend upon **Terraform resource dependencies**
9. Ensure IaC is **idempotent**
10. **Treat your infrastructure like an application** (change control process, testing, monitoring)
Infrastructure and k8s-repo
Infrastructure Repo

Infrastructure

- team
- environment
- GCP resource

network

prod

host_project

main.tf
variables.tf
script.sh

main.tf
variables.tf
script.sh
k8s-repo Repo

```
+-----------------+                  +-----------------+
| k8s-repo        |                  | k8s-repo        |
|                 |                  |                 |
| cluster         |                  | gke-1-app1-r1a-prod |
|                 |                  |                 |
| resource        |                  | app             |
|                 |                  | deployments     |
| sub resource    |                  |                 |
|                 | YAMLs             |                 |
|                 | kustomization.yaml|                 |
|                 | YAMLs             |                 |
|                 | kustomization.yaml|                 |
|
```
Lab 7: Deploy sample app

Objective: Deploy Hipster shop app on apps clusters

- Clone k8s-repo
- Copy Hipster shop manifests to all apps clusters
- Create Services for Hipster shop app in the ops clusters
- Setup loadgenerators in the ops clusters to test global connectivity
- Verify secure connectivity to the Hipster shop app

30 mins
Distributed Services
Multi-cluster ⇒ Distributed Services

microservices + multi-cluster = Distributed Services

A Distributed Service is a collection of k8s services across multiple clusters.

globally available
resilient
scalable
flexible + reliable software delivery
Internet

HTTP

Frontend

Product CatalogService
Internet (GCLB)

HTTP

Frontend

Product CatalogService

Frontend

Product CatalogService

Frontend

Product CatalogService

Frontend

Product CatalogService
Internet (GCLB)

HTTP

Frontend

Product CatalogService

Product CatalogService

Product CatalogService

Product CatalogService
<table>
<thead>
<tr>
<th>service name:</th>
<th>serving from:</th>
</tr>
</thead>
<tbody>
<tr>
<td>frontend</td>
<td>r1-a, r1-b, r2-b, r2-c</td>
</tr>
<tr>
<td>productcatalog</td>
<td>r1-a, r1-b, r2-b, r2-c</td>
</tr>
</tbody>
</table>
Consider a bad productcatalog rollout introduces a bug in one zone...

- SLO alert fires,
- SRE drains that zone,
- halts the rollout
- investigates,
- rollback or: fix and roll forward
- undrain

<table>
<thead>
<tr>
<th>service name</th>
<th>serving from:</th>
<th>SLO</th>
<th>rollout</th>
<th>drains</th>
</tr>
</thead>
<tbody>
<tr>
<td>frontend</td>
<td>r1-a, r1-b, r2-b, r2-c</td>
<td>99.96%</td>
<td>DONE</td>
<td>none</td>
</tr>
<tr>
<td>productcatalog</td>
<td>r1-a, r1-b, r2-b, r2-c</td>
<td>97.4%</td>
<td>PAUSED</td>
<td>r2-b</td>
</tr>
</tbody>
</table>
Thinking in terms of Distributed Services

- Observability (SLOs)
- Capacity Planning (N+2)
- Rollouts
- Draining, Spilling
- Rollbacks
- Disaster Recovery
Distributed Services best practices and lessons learned

1. Multicluster lets you think of your services **across clusters**.
2. Trying to keep track of every pod to/from every other service's pods is **complicated**!
3. Simplify by thinking about **Services** at a "global" level.
4. Define and track your SLOs, Rollouts, Policies, Traffic management rules **per service**
5. Service Owning teams operate **independent** from each other, with common tools
Observability
Observing Distributed Systems

- Host-based Monitoring
- Log grepping
- "Tweaking Thresholds"
- Blackbox
- Cause-based Alerts

- Instrumented Code
- Metrics Aggregation
- Structured Logs
- Distributed Traces
- DevOps

- SLOs / Error Budgets
- Symptom-based Alerts
- Mesh-Driven
- Inter-node metrics
- Service Ownership

- Raw Events
- Auto Instrumentation
- High Cardinality Fields
- O-driven development
## Separate Needs

### Service Health / Performance
- **SLOs:** Availability, Latency
- **MTTD, MTTR**
- **MTLS, Service Authz**
- **Capacity (N+2)**
- **Deploy Lead Time**

### Mesh Health/Performance
- **Pilot, Mixer SLOs**
- **Loadbalancer stats, config**
- **Policy Delivery/Converge Time**
- **Managed Storage: sharding, failover**

### Infra Health/Performance
- **Infra Health/Performance**

### Host-level Metrics:
- **cpu_load, disk_free, num_threads**

### Host-level Metrics Details:
- **VM Metrics**
- **Container Metrics**
- **Network Metrics**
- **Storage Metrics**
Common Tools

**Service Health / Performance**
- Monitoring
- Logging
- Trace
- Debugger
- Error Reporting

**Mesh Health/Performance**
- Monitoring
- Logging
- Cloud Service Mesh
- Prometheus
- Grafana

**Infra Health/Performance**
- Monitoring
- Logging

**Host, Network**
- Monitoring
- Logging
Metrics

Graphs

Metric Name: "bytes received"

Resource Type: "GKE Container" or "HTTP LB"

Filter by: "app=frontend", "env=staging"

Group by: "cluster", "env", "service"

Aggregator (mean, max, sum, ...)

Dashboards

Sets of Graphs
Aligned to Timespan
Filter all Graphs
Logging

Showing logs from the last hour ending at 4:00 PM (PST)

- 2020-02-12 16:00:04.910 PST  ...rencyservice-server received conversion request
- 2020-02-12 16:00:04.910 PST  ...rencyservice-server conversion request successful
- 2020-02-12 16:00:04.912 PST  ...rencyservice-server received conversion request
- 2020-02-12 16:00:04.913 PST  ...rencyservice-server conversion request successful

Filter by label or text search

Kubernetes Container  All logs  Any log level  Last hour

Convert to advanced filter
Get link to filter

1. resource.type="k8s_container"
2. jsonPayload.name="currencyserver-service"
Logging

Log Resource Type:

<table>
<thead>
<tr>
<th>Resource type</th>
<th>Display name</th>
</tr>
</thead>
<tbody>
<tr>
<td>k8s_cluster</td>
<td>Kubernetes Cluster Audit Log</td>
</tr>
<tr>
<td>k8s_container</td>
<td>(don't use gke_container)</td>
</tr>
<tr>
<td>gke_container</td>
<td>(old)</td>
</tr>
</tbody>
</table>

```json
{
    "insertId": "zb6ac40j82ulgnz",
    "jsonPayload": {
        "hostname": "currency-service-666f85df9-zzbz5",
        "message": "GETTING_SUPPORTED_CURRENCIES...",
        "name": "currency-service-server",
        "pid": 1,
        "v": 1
    }
}
```

```json
labels: {
    "k8s-pod/app": "currency-service",
    "k8s-pod/app_kubernetes.io/managed-by": "skaffold=v0.38.0",
    "k8s-pod/template-hash": "666f85df9",
    "k8s-pod/skaffold_dev/builder": "local",
    "k8s-pod/skaffold_dev/cleanup": "true",
    "k8s-pod/skaffold_dev/deployer": "kubectl",
    "k8s-pod/skaffold_dev/docker-api-version": "1.40",
    "k8s-pod/skaffold_dev/run-id": "ce3935be-c9be-4f6c-8e82-f251350740c",
    "k8s-pod/skaffold_dev/tag-policy": "git-commit",
    "k8s-pod/skaffold_dev/tail": "true"
}
```

```json
logName: "projects/rubbo-istio1/logs/stdout"
receiveTimestamp: "2020-02-12T23:56:15.362264557Z"
```

```json
resource: {
    "labels": {
        "cluster_name": "central",
        "container_name": "server",
        "location": "us-east1-b",
        "namespace_name": "default",
        "pod_name": "currency-service-666f85df9-zzbz5",
        "project_id": "rubbo-istio1"
    }
}
```

```json
severity: "INFO"
timestamp: "2020-02-12T23:56:12.629748664Z"
```
Istio telemetry

metrics, logs and traces, gathered by sidecars

Stackdriver: Metrics, Dashboards, Traces, Logs
Observability best practices and lessons learned

1. Encode graphs and **dashboards in code**, in your repo, share and review across teams.
2. Familiarize Devs with **Distributed Traces**. Adding more code-level Instrumentation helps!
3. Use a centralized logging system to correlate logs across multiple sources.
4. Let Istio provide observability between services.
5. As you gain comfort with Observability, **add and improve instrumentation** over time.
Lab 8: Observability with Stackdriver

Objective: Connect Istio telemetry to Stackdriver and validate.

- Install `istio-telemetry` resources
- Create/update Services dashboards using Istio metrics
- View container logs
- View distributed tracing in Stackdriver

20 mins
Multi cluster service discovery
shared-vpc

**project-2-dev1-apps**

- subnet-3 with secondary ranges in region 1
  - gke-2-apps-r1b-prod (region-1-zone-b, apps)
    - Hipster Shop
  - gke-1-apps-r1a-prod (region-1-zone-a, apps)
    - Hipster Shop

**project-3-dev2-apps**

- subnet-4 with secondary ranges in region 2
  - gke-4-apps-r2b-prod (region-2-zone-b, apps)
  - gke-3-apps-r2a-prod (region-2-zone-c, apps)

**project-1-ops-asm**

- subnet-1 with secondary ranges in region 1
  - gke-asm-1-r1-prod (region-1-regional, controlplane)
    - istio install (multicluster + mesh expansion, mTLS)

**project-3-dev2-apps**

- subnet-2 with secondary ranges in region 2
  - gke-asm-2-r2-prod (region-2-regional, controlplane)
    - istio install (multicluster + mesh expansion, mTLS)
shared-vpc

- gke-2-apps-r1b-prod (region-1-zone-b, apps)
- gke-1-apps-r1a-prod (region-1-zone-a, apps)
- gke-3-apps-r2a-prod (region-2-zone-c, apps)
- gke-4-apps-r2b-prod (region-2-zone-b, apps)
- gke-asm-1-r1-prod (region-1-regional, controlplane)
- gke-asm-2-r2-prod (region-2-regional, controlplane)

istio install (multicluster + mesh expansion, mTLS)
istio install (multicluster + mesh expansion, mTLS)
shared-vpc

- gke-2-apps-r1b-prod (region-1-zone-b, apps)
  - Pod
- gke-1-apps-r1a-prod (region-1-zone-a, apps)
  - Pod
- gke-2-apps-r1b-prod (region-1-zone-b, apps)
  - Pod
- gke-4-apps-r2b-prod (region-2-zone-b, apps)
  - Pod
- gke-3-apps-r2a-prod (region-2-zone-c, apps)
  - Pod
- gke-asm-1-r1-prod (region-1-regional, controlplane)
  - istio-telemetry
  - pilot
  - istio-policy
  - istio install (multicluster + mesh expansion, mTLS)
- gke-asm-2-r2-prod (region-2-regional, controlplane)
  - istio-telemetry
  - pilot
  - istio-policy
  - istio install (multicluster + mesh expansion, mTLS)
shared-vpc

- gke-2-apps-r1b-prod (region-1-zone-b, apps)
- gke-1-apps-r1a-prod (region-1-zone-a, apps)
- gke-4-apps-r2b-prod (region-2-zone-b, apps)
- gke-3-apps-r2a-prod (region-2-zone-c, apps)
- gke-asm-1-r1-prod (region-1-regional, controlplane)
- gke-asm-2-r2-prod (region-2-regional, controlplane)

**istio install (multicluster + mesh expansion, mTLS)**
istio install (multicluster + mesh expansion, mTLS)
Multi cluster best practices and lessons learned

1. Use shared VPC
2. Use VPC native GKE clusters
3. Create every Service in all clusters
4. Create kubeconfig secrets for every cluster in the ASM control plane clusters
5. Create ServiceEntries for out-of-VPC services
Certificates and mTLS
root

offline

long lived
typically 10 yrs
root

intermediate

offline

online

long lived typically 10 yrs

medium life typically 1 yr
root
- offline
  - long lived
    - typically 10 yrs

intermediate
- online
  - medium life
    - typically 1 yr

leaf
- non-signing
  - short lived
    - hours or days
- **root CA**: Sign

- **citadel**: Sign

- **envoy**: Sign

- **root**: Sign

- **intermediate**: Sign

- **leaf**: Non-signing

- **offline**: Long lived, typically 10 yrs

- **online**: Medium life, typically 1 yr

- **non-signing**: Short lived, hours or days
shared-vpc

- gke-2-apps-r1b-prod (region-1-zone-b, apps)
  - citadel
  - Pod
- gke-1-apps-r1a-prod (region-1-zone-a, apps)
  - citadel
  - Pod
- gke-3-apps-r2a-prod (region-2-zone-c, apps)
  - citadel
  - Pod
- gke-4-apps-r2b-prod (region-2-zone-b, apps)
  - citadel
  - Pod

- gke-asm-1-r1-prod (region-1-regional, controlplane)
  - citadel
  - istio install (multicluster + mesh expansion, mTLS)
- gke-asm-2-r2-prod (region-2-regional, controlplane)
  - citadel
  - istio install (multicluster + mesh expansion, mTLS)
shared-vpc

makecerts.sh

root CA

```
gke-2-apps-r1b-prod (region-1-zone-b, apps)
gke-1-apps-r1a-prod (region-1-zone-a, apps)
gke-asm-1-r1-prod (region-1-regional, controlplane)
gke-3-apps-r2a-prod (region-2-zone-c, apps)
gke-4-apps-r2b-prod (region-2-zone-b, apps)
gke-asm-2-r2-prod (region-2-regional, controlplane)
```

Pod

Pod

Pod

Pod

Pod

```
istio install (multicluster + mesh expansion, mTLS)
```

citadel

citadel

citadel

citadel

citadel

citadel

istio install (multicluster + mesh expansion, mTLS)
gke-2-apps-r1b-prod (region-1-zone-b, apps)

namespace frontend

gke-1-apps-r1a-prod (region-1-zone-a, apps)

gke-asm-1-r1-prod (region-1-regional, controlplane)

gke-3-apps-r2a-prod (region-2-zone-c, apps)

gke-4-apps-r2b-prod (region-2-zone-b, apps)

gke-asm-2-r2-prod (region-2-regional, controlplane)

istio install (multicluster + mesh expansion, mTLS)

root CA

makecerts.sh

Pod

Pod

Pod

Pod

Pod

Pod

Pod

Pod

citadel

citadel

citadel

citadel

citadel

citadel

citadel

citadel

citadel

sa certs

secret

root CA

istio install (multicluster + mesh expansion, mTLS)
Lab 9: Mutual TLS

Objective: Secure connectivity between microservices (AuthN).

- Enable mesh wide mTLS
- Verify mTLS

20 mins
Canary Deployments with Istio
Deployment and Testing Strategies

Red/black
(Blue/green)

Rolling red/black

Canary
Canary Deployments on Istio

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
namespace: frontend
spec:
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: server
          image: image1
```

Frontend production deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-v2
namespace: frontend
spec:
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: server
          image: image1
```

Frontend baseline deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-v2
namespace: frontend
spec:
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: server
          image: image1
```

Frontend canary deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-v2
namespace: frontend
spec:
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: server
          image: image2
```

Frontend canary deployment
Canary Deployments on **Istio**

```yaml
apiVersion: "networking.istio.io/v1alpha3"
kind: "DestinationRule"
metadata:
  name: "frontend"
  namespace: "frontend"
spec:
  host: "frontend.frontend.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2

apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: frontend
  namespace: frontend
spec:
  hosts:
  - frontend
  http:
    - route:
      - destination:
        host: frontend
        subset: v1
        weight: 100
      - destination:
        host: frontend
        subset: v2
        weight: 0
```
Deployment best practices and lessons learned

1. Use Continuous integration/Continuous deployment (CI/CD) methodologies.
2. Automation is key
3. Deploy to one region/cluster at a time to avoid global outages
4. Have a multicloud rollout strategy
5. Have a rollback strategy
6. Post-deployment monitoring
Lab 10: Canary Deployments
Objective: Rollout new version of the frontend Service.
- Rollout frontend-v2 (next production ver) Service in one region
- Use DestinationRules and VirtualServices to slowly steer traffic to frontend-v2
- Verify GitOps deployment pipeline

Lab 11: Authorization Policy
Objective: Set up RBAC between microservices (AuthZ).
- Create AuthorizationPolicy to DENY access to a microservice
- Create AuthorizationPolicy to ALLOW specific access to a microservice

60 mins
Multi-Cluster Ingress
Anatomy of a load balancer
Managed **SSL Certificates**

- Automatically create and renew Let's Encrypt certificates
- Certificates can be used with HTTPS target proxies
- Requires domain name that points to global address
Cloud Endpoints for DNS

Conventional project scoped DNS records in the form of:

<name>.endpoints.<project>.cloud.goog

Create endpoints service with minimal spec and the following fields:

host: "<name>.endpoints.<project>.cloud.goog"
x-google-endpoints:
  - name: "<name>.endpoints.<project>.cloud.goog"
  target: "<ip address>"
Network Endpoint Groups

- VPC-Native Clusters
- NEG Controller
- Automatic Backend Service Updates
Istio-ingressgateway service

```yaml
apiVersion: v1
class: Service
metadata:
  annotations:
    cloud.google.com/neg: '{}"exposed_ports": {"80":{}}'
    anthos.cft.dev/autoneg: '{}"name":"istio-ingressgateway", "max_rate_per_endpoint":100}'
spec: ...
```

NEG controller (managed by GKE)

autoNEG controller (self managed)
Istio-ingressgateway service

```yaml
apiVersion: v1
kind: Service
metadata:
  annotations:
    cloud.google.com/neg: '{"exposed_ports": {"80":{}}}'
    anthos.cft.dev/autoneg: '{"name":"istio-ingressgateway", "max_rate_per_endpoint":100}"

spec: ...
```

**neg controller**

**autoneg controller**
How is a Distributed Service built?
Properties of a **Distributed Service**:

- **Is observable** (aggregated metrics, distributed traces)
- **Is resilient** to localized failure (per cluster, deployment, circuit breaking, fault injection)
- Allows for **gradual change** (eg: canary traffic)
- **Is secure** (eg: least privilege, encrypted communications)
- Can **enforce policies** globally or per-service (eg: MTLS, HTTP actions)
- Can be **controlled** quickly, locally (immediate cluster drain, rollback, rate limiting)
Distributed Service

Kubernetes
Distributed Service

PodSecurityPolicy

Pod spec admission controller

Kubernetes

K8s hardening
Distributed Service

- RBAC (Roles and Bindings)
- Workload Identity
- Kubernetes Service Account
- Namespace
- PodSecurityPolicy

What a KSA can do, at cluster or namespace level - Authz
- Use GCP Svc Acct to access GCP resources e.g StackDriver
- Identity of a service
- Virtual cluster
- Pod spec admission controller

K8s identity and authz

K8s hardening
Distributed Service

- Service
- Deployment
- RBAC (Roles and Bindings)
- Workload Identity
- Kubernetes Service Account
- Namespace
- PodSecurityPolicy

- VIP/DNS name over a Deployment’s replica Pods
- Pod templates to create replicas
- What a KSA can do, at cluster or namespace level - Authz
- Identity of a service
- Virtual cluster
- Pod spec admission controller

- K8s workload definitions
- K8s identity and authz
- K8s hardening
<table>
<thead>
<tr>
<th>Kubernetes</th>
<th>Distributed Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>VirtualService</td>
<td>Name based routing, canary rollouts</td>
</tr>
<tr>
<td>DestinationRule</td>
<td>Subsets, LB policy, mTLS, circuit breakers</td>
</tr>
<tr>
<td>AuthorizationPolicy</td>
<td>Allowed actions to a Service from SAs - Authz</td>
</tr>
<tr>
<td>MeshPolicy / Policy</td>
<td>mTLS, either at mesh level or per Service level - Authn</td>
</tr>
<tr>
<td>Service</td>
<td>VIP/DNS name over a Deployment’s replica Pods</td>
</tr>
<tr>
<td>Deployment</td>
<td>Pod templates to create replicas</td>
</tr>
<tr>
<td>RBAC (Roles and Bindings)</td>
<td>What a KSA can do, at cluster or namespace level - Authz</td>
</tr>
<tr>
<td>Workload Identity</td>
<td>Use GCP Svc Acct to access GCP resources e.g StackDriver</td>
</tr>
<tr>
<td>Kubernetes Service Account</td>
<td>Identity of a service</td>
</tr>
<tr>
<td>Namespace</td>
<td>Virtual cluster</td>
</tr>
<tr>
<td>PodSecurityPolicy</td>
<td>Pod spec admission controller</td>
</tr>
</tbody>
</table>

**Istio CRs**

**K8s workload definitions**

**K8s identity and authz**

**K8s hardening**
kind: Namespace
name: currency
labels:
  istio-injection: enabled
```yaml
kind: ServiceAccount
metadata:
  name: currency
  namespace: currency
annotations:
  iam.gke.io/gcp-service-account: microservices@user309-200224-01-dev1-o52jm2.iam.gserviceaccount.com

kind: Namespace
name: currency
labels:
  istio-injection: enabled
```
kind: PodSecurityPolicy
ame: currency
privileged: false # Prevents creation of privileged Pods

kind: ServiceAccount
metadata:
  name: currency
  namespace: currency
annotations:
    iam.gke.io/gcp-service-account: microservices@user309-200224-01-dev1-o52jm2.iam.gserviceaccount.com

kind: Namespace
name: currency
labels:
  istio-injection: enabled
kind: Role
rules:
  - apiGroups:
    - extensions
    resources:
    - podsecuritypolicies
    verbs:
    - use
    resourceNames:
    - currency

kind: PodSecurityPolicy
name: currency
privileged: false # Prevents creation of privileged Pods

kind: ServiceAccount
metadata:
  name: currency
namespace: currency
annotations:
  iam.gke.io/gcp-service-account: microservices@user309-200224-01-dev1-o52jm2.iam.gserviceaccount.com

kind: Namespace
name: currency
labels:
  istio-injection: enabled
kind: RoleBinding
roleRef:
  kind: ClusterRole
  name: currency
  apiGroup: rbac.authorization.k8s.io
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:serviceaccounts

kind: Role
rules:
  - apiGroups:
      - extensions
    resources:
      - podsecuritypolicies
    verbs:
      - use
    resourceNames:
      - currency

kind: PodSecurityPolicy
name: currency
privileged: false # Prevents creation of privileged Pods

kind: ServiceAccount
metadata:
  name: currency
  namespace: currency
  annotations:
    iam.gke.io/gcp-service-account: microservices@user309-200224-01-dev1-o52jm2.iam.gserviceaccount.com

kind: Namespace
name: currency
labels:
  istio-injection: enabled
kind: Deployment
image: gcr.io/google-samples/microservices-demo/currencyservice:v0.1.3

kind: RoleBinding
roleRef:
  kind: ClusterRole
  name: currency
  apiGroup: rbac.authorization.k8s.io
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:serviceaccounts

kind: Role
rules:
  - apiGroups:
      - extensions
    resources:
      - podsecuritypolicies
    verbs:
      - use
      resourceNames:
        - currency

kind: PodSecurityPolicy
name: currency
privileged: false # Prevents creation of privileged Pods

kind: ServiceAccount
metadata:
  name: currency
  namespace: currency
annotations:
  iam.gke.io/gcp-service-account: microservices@user309-200224-01-dev1-o52jm2.iam.gserviceaccount.com

kind: Namespace
name: currency
labels:
  istio-injection: enabled
kind: Deployment
image: gcr.io/google-samples/microservices-demo/currencyservice:v0.1.3

kind: RoleBinding
roleRef:
  kind: ClusterRole
  name: currency
  apiGroup: rbac.authorization.k8s.io

subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:serviceaccounts

kind: Role
rules:
- apiGroups:
  - extensions
  resources:
  - podsecuritypolicies
  verbs:
  - use
  resourceNames:
  - currency

kind: PodSecurityPolicy
name: currency
privileged: false # Prevents creation of privileged Pods

kind: ServiceAccount
metadata:
  name: currency
  namespace: currency
  annotations:
    iam.gke.io/gcp-service-account: microservices@user309-200224-01-dev1-o52jm2.iam.gserviceaccount.com

kind: Service
spec:
  type: ClusterIP
  selector:
    app: currencyservice
  ports:
  - name: grpc
    port: 7000
**Namespace labels:**

- istio-injection: enabled

**Kind: Role**

rules:
- apiGroups:
  - extensions
  - podsecuritypolicies
verbs:
- use
resourceNames:
- currency

**Kind: RoleBinding**

roleRef:
  kind: ClusterRole
  name: currency
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:serviceaccounts

**Kind: Role**

rules:
- apiGroups: extensions
  resources: podsecuritypolicies
  verbs: use
  resourceNames: currency

**Kind: PodSecurityPolicy**

name: currency
privileged: false # Prevents creation of privileged Pods

**Kind: Deployment**

image: gcr.io/google-samples/microservices-demo/currencyservice:v0.1.3

**Kind: Service**

spec:
  type: ClusterIP
  selector:
    app: currencyservice
  ports:
    - name: grpc
      port: 7000

**Kind: AuthorizationPolicy**

spec:
  selector:
    matchLabels:
      app: currencyservice
  rules:
    - from:
      - source:
        principals: 
        - "cluster.local/ns/frontend/sa/frontend"

**Kind: ServiceAccount**

metadata:
  name: currency
namespace: currency
annotations:
  iam.gke.io/gcp-service-account: microservices@user309-200224-01-dev1-o52jm2.iam.gserviceaccount.com

**Kind: Namespace**

name: currency
labels:
  istio-injection: enabled
kind: Deployment
text: gcr.io/google-samples/microservices-demo/currencyservice:v0.1.3

kind: RoleBinding
roleRef:
  kind: ClusterRole
  name: currency
  apiGroup: rbac.authorization.k8s.io
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:serviceaccounts

kind: Role
rules:
  - apiGroups:
      - extensions
        resources:
          - podsecuritypolicies
        verbs:
          - use
        resourceNames:
          - currency

kind: PodSecurityPolicy
name: currency
privileged: false # Prevents creation of privileged Pods

kind: ServiceAccount
metadata:
  name: currency
namespace: currency
annotations:
  iam.gke.io/gcp-service-account: microservices@user309-200224-01-dev1-o52jm2.iam.gserviceaccount.com

kind: Namespace
name: currency
labels:
  istio-injection: enabled

Kind: DestinationRule
spec:
  host: "frontend.frontend.svc.cluster.local"
trafficPolicy:
  tls:
    mode: ISTIO_MUTUAL
subsets:
  - name: v2
    labels:
      version: v2

kind: AuthorizationPolicy
spec:
  selector:
    matchLabels:
      app: currencyservice
  rules:
    - from:
      - source:
        principals: ["cluster.local/ns/frontend/sa/frontend"]

kind: Service
spec:
  type: ClusterIP
  selector:
    app: currencyservice
  ports:
    - name: grpc
      port: 7000
kind: Deployment
image: gcr.io/google-samples/microservices-demo/currencyservice:v0.1.3

kind: RoleBinding
roleRef:
  kind: ClusterRole
  name: currency
apiGroup: rbac.authorization.k8s.io
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:serviceaccounts

kind: PodSecurityPolicy
name: currency
privileged: false # Prevents creation of privileged Pods

kind: ServiceAccount
metadata:
  name: currency
namespace: currency
annotations:
  iam.gke.io/gcp-service-account: microservices@user309-200224-01-dev1-o52jm2.iam.gserviceaccount.com

kind: Namespace
name: currency
labels:
  istio-injection: enabled

kind: Service
spec:
type: ClusterIP
selector:
  app: currencyservice
ports:
  - name: grpc
    port: 7000

kind: VirtualService
spec:
  hosts:
  - frontend
  http:
    - route:
      - host: frontend
        subset: v2
        port:
          number: 80
          weight: 100

Kind: DestinationRule
spec:
  host: "frontend.frontend.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: v2
    labels:
      version: v2

kind: AuthorizationPolicy
spec:
  selector:
    matchLabels:
      app: currencyservice
  rules:
  - from:
    - source:
      principals: ["cluster.local/ns/frontend/sa/frontend"]
Properties of a Distributed Service:

- **observable** - istio-telemetry
- **resilient** - DestinationRule (circuit breaker)
- **gradual change** - VirtualService + DestinationRule
- **secure** - MeshPolicy + Policy
- **enforce policies** - AuthorizationPolicy
- **controlled quickly** - VirtualService
Kubernetes Service best practices and lessons learned

1. **Use** PodSecurityPolicy
2. **Separate namespaces and service accounts for Services**
3. **Use RBAC** (required if using PodSecurityPolicy)
4. **Use Workload Identity (GCP SA) to access GCP resources** (GKE only)
   a. **Principle of least privilege**
5. **Define a VirtualService and DestinationRule for every service** (ASM/Istio only)
   a. App rollout
   b. App migration
   c. Authn (mTLS)
   d. Circuit Breakers
   e. DNS
6. **Define an AuthorizationPolicy per service** (ASM/Istio only)
   a. Privileged Services
Lab 12: Infrastructure Scaling

Objective: Scale infrastructure by adding new region, project, and clusters.

- Clone the `infrastructure` repo
- Update the terraform files to create new resources
  - 2 subnets in the new region (one for the ops project and one for the new project)
  - New ops cluster in new region (in the new subnet)
  - New Istio control plane for the new region
  - 2 apps clusters in the new project in the new region
- Commit to `infrastructure` repo
- Verify installation

30 mins
What’s next?

1. Build the **infrastructure** in your organization
2. Deploy a **Distributed Service**
3. Observe, Secure, Traffic Mgt
4. Call us. **Let’s build together**
Or for more specific business needs, you can...

1. Request an exploratory discovery session with our hybrid modernisation specialists to kick start your hybrid cloud journey

2. Book in an Anthos or ASM hands-on workshop, adapted for your business, on your premise

3. Request a hybrid modernisation review by Google Cloud specialists on a specific workload for your business

Just tick the box in the Feedback form and your Account Owner will be in touch.... No spam!
Lab 13: Circuit Breaking
Objective: Create a resilient platform by implementing a Circuit Breaker for the shipping Service.

- Create a DestinationRule for the shipping Service to implement a circuit breaker
- Use fortio (a load gen utility) to validate circuit breaker for the shipping Service by force tripping the circuit

Lab 14: Fault injection
Objective: Test the resiliency of the recommendation Service by introducing delays (before it is pushed to production).

- Create a VirtualService for the recommendation Service to introduce a 5s delay
- Test the delay using fortio load generator
- Remove the delay in the VirtualService and validate
Thank you