Microsoft

# Kubernetes on Azure


Scott Coulton
*Developer Advocate*

@scottcoulton

# About me.

Scott Coulton

Developer Advocate

Spent the last 4 years on container related development

I love golang

I am also a Docker Captain



@scottcoulton

scotty-c

# Agenda

- Kubernetes 101
  - Introduction into Kubernetes
  - Kubernetes components
  - Deploying Kubernetes on Azure
  - Pods, services and deployments
  - Rabc, roles and service accounts
  - Stateful sets
  - Kubernetes networking and service discovery
  - Load balancing and ingress control

@scottcoulton

# Agenda

- Helm
  - Introduction into Helm
  - Understanding charts
  - Deploying Helm on Kubernetes
  - Helm cli
  - Deploying a public chart
  - Writing our own chart
  - Helm and CNAB

# Agenda

- Kubernetes advanced topics
  - Virtual node with virtual kubelet
  - Pod security context
  - Introduction to istio
  - Advanced application routing with istio
  - Setting mTLS between application services with istio

# Course assumptions

## Prior knowledge

- A basic understanding of Linux
- Be able to read bash scripts
- Understand what a container is

## Equipment needed

- A bash shell (WSL is fine)
- An Azure account with access to create resources service principals
- Azure cli 2.0


@scottcoulton

# Tools we will need

## Please install

- kubectl https://kubernetes.io/docs/tasks/tools/install-kubectl/
- kubectx https://github.com/ahmetb/kubectx
- jq (from your package manager)


@scottcoulton

# Code examples

Code for this course can be downloaded from

https://github.com/scotty-c/kubernetes-on-azure-workshop

# Introduction into Kubernetes


@scottcoulton

# So what is Kubernetes

Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available. Google open-sourced the Kubernetes project in 2014.

# Why do I need Kubernetes and what can it do?

Kubernetes has a number of features. It can be thought of as:

- a container platform
- a microservices platform
- a portable cloud platform and a lot more

# Why do I need Kubernetes and what can it do?

Kubernetes provides a **container-centric** management environment. It orchestrates computing, networking, and storage infrastructure on behalf of user workloads. This provides much of the simplicity of Platform as a Service (PaaS) with the flexibility of Infrastructure as a Service (IaaS), and enables portability across infrastructure providers.

# Kubernetes components


@scottcoulton

# Kubernetes components

Kubernetes is broken down into two node types.

- Master node
- Worker node

# Master node

A master node is responsible for

- Running the control plane
- Scheduling workloads
- Security controls

# Worker node

A worker node is responsible for

- Running workloads

# Master node

A master nodes components (control plane)

- kube-apiserver
- etcd
- kube-scheduler
- kube-controller-manager
- cloud-controller-manager

# Worker node

A worker nodes components

- kubelet
- Kube-proxy

# Kube-apiserver

The kube-apiserver is responsible for

- The entry point into the cluster
- It exposes the Kubernetes API
- It's a REST service
- Validates and configures data for the api objects

# etcd

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data

Exciting news Cosmos DB has an etcd api

# kube-scheduler

Kube-scheduler is responsible for

- watches newly created pods that have no node assigned, and selects a node for them to run on

Factors taken into account for scheduling decisions include individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference and deadlines

# kube-controller-manager

## Kube-controller-manager is responsible for

- Node Controller: Responsible for noticing and responding when nodes go down.
- Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
- Endpoints Controller: Populates the Endpoints object (that is, joins Services & Pods)
- Service Account & Token Controllers: Create default accounts and API access tokens for new namespaces.

# cloud-controller-manager

## Cloud-controller-manager is responsible for

- For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- For setting up routes in the underlying cloud infrastructure
- For creating, updating and deleting cloud provider load balancers
- For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes

# kubelet

## Kubelet is responsible for

- All containers in a pod are running

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy

# Kube-proxy

This reflects services as defined in the Kubernetes API on each node and can do simple TCP, UDP, and SCTP stream forwarding or round robin TCP, UDP, and SCTP forwarding across a set of backends

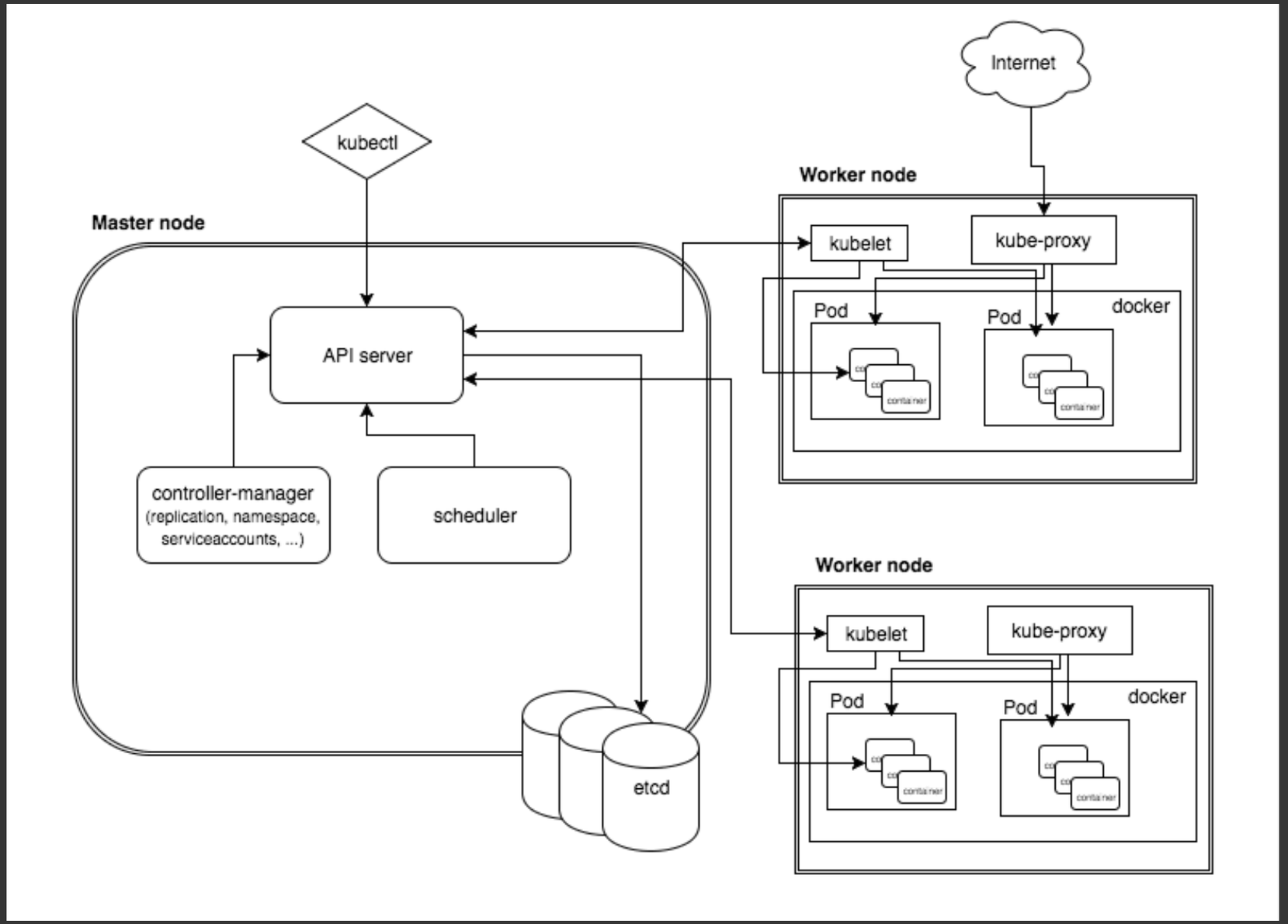
# Container runtimes

Kubernetes can use differnet container runtimes

- Docker
- Moby
- Containerd
- Cri-o

At Azure we use Moby

# Kubernetes architecture



# Deploying Kubernetes on Azure


@scottcoulton

# The offical docs are here

https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough

@scottcoulton

# Create a resource group

```
az group create --name k8s --location eastus
```

# Create your cluster

```
az aks create --resource-group k8s \
    --name k8s \
    --generate-ssh-keys \
    --kubernetes-version 1.12.5 \
    --enable-rbac \
    --node-vm-size Standard_DS2_v2
```

# If you don’t have kubectl already

```
az aks install-cli
```

# Get cluster credentials

```
az aks get-credentials --resource-group k8s --name k8s
```

# Test out your cluster

```
kubectl get nodes

kubectl get pods --all-namespaces
```

# Pods, services and deployments

@scottcoulton

"A pod is not equal to a container"

@scottcoulton

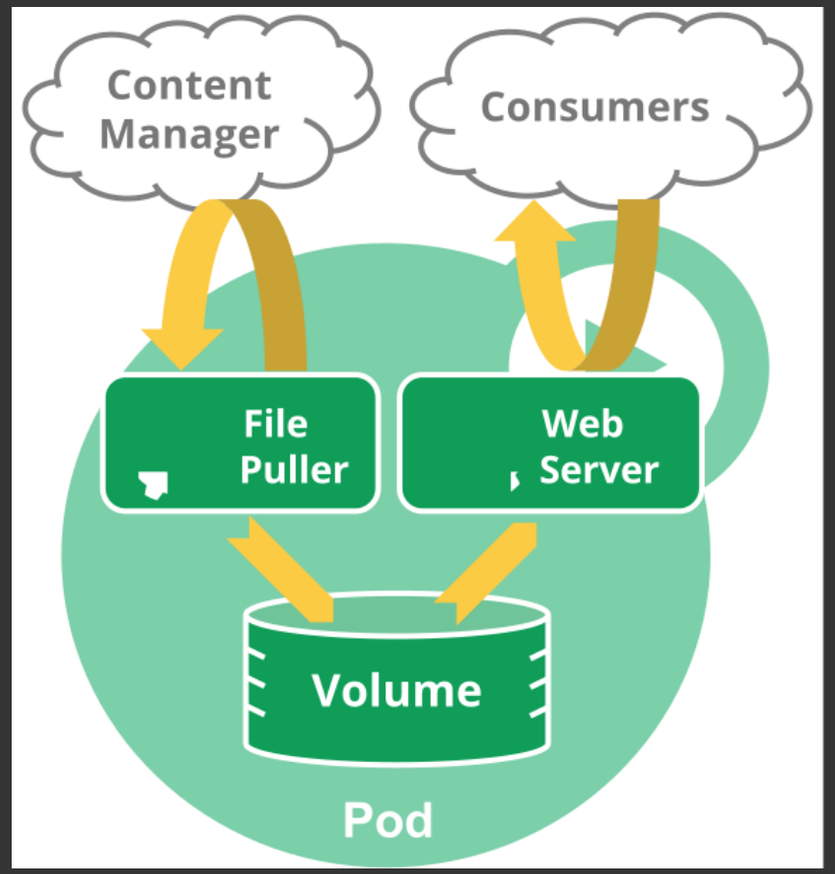
# Pods

Pods are a single or group of containers that share

- localhost
- storage
- ip address
- port range

The shared context of a pod is a set of Linux namespaces, cgroups, and potentially other facets of isolation

# Pods


Content Manager
Consumers
File Puller
Web Server
Volume
Pod

# Defining a pod

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! &&
sleep 3600']
```

# Services

A service in Kubernetes exposes a set of pods

- Creates a vip
- Sets up basic routing to the pods
- Talks to the cloud-manager to assign a public ip or load balancer

# Defining a service

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

# Replicaset

A replicaset defines that state of a running application

- The amount of pods that should are running
- It self-heals the pods to their disered state

# Deployments

A deployments defines the lifecycle of an application

- Is made up of pods
- It controls replicasets
- Includes the functionality to update the desired state
- Rolling updates are included

# Defining a deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

# Let's deploy our own deployment

@scottcoulton

# Our deployment

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp-deployment
spec:
  selector:
    matchLabels:
      app: webapp
  replicas: 3
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: scottyc/webapp:latest
        ports:
        - containerPort: 3000
          hostPort: 3000
EOF
```

# Check our deployment

```
kubectl get deployments
```

```
kubectl get pods
```

```
kubectl get service
```

# What issue did you find ?


@scottcoulton

# Answer: We have not exposed a service


@scottcoulton

# Exposing our deployment

```
kubectl expose deployment webapp-deployment --
type=LoadBalancer
```

```
kubectl get service
```

# To access our app http://<your-pub-ip>:3000


@scottcoulton

# Rbac, roles and service accounts

@scottcoulton

# rbac

Role based access control (rbac)

- Seperation of applications
- Access control for users
- Access control for applications (service accounts)

# namespaces

Namespaces are the logical serperation in kubernetes
Things that are namespaced

- dns <service-name>.<namespace-name>.svc.cluster.local
- Deployments, services and pods
- Access control for applications (service accounts)
- Resource quotas
- Secrets

# namespaces

Things that are NOT namespaced

- Nodes
- Networking
- Storage

# Service accounts vs user accounts

The differences are

- User accounts are for humans. Service accounts are for processes, which run in pods.
- User accounts are intended to be global. Names must be unique across all namespaces of a cluster, future user resource will not be namespaced. Service accounts are namespaced.

# Create a namespace

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: webapp-namespace
EOF
```

# Create a service account

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: webapp-service-account
  namespace: webapp-namespace
EOF
```

# Create a role

```
cat <<EOF | kubectl apply -f -
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: webapp-role
  namespace: webapp-namespace
rules:
  - apiGroups: [""]
    resources: ["pods", "pods/log"]
    verbs: ["get", "list", "watch"]
EOF
```

# Create a role binding

```
cat <<EOF | kubectl apply -f -
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: webapp-role-binding
  namespace: webapp-namespace
subjects:
  - kind: ServiceAccount
    name: webapp-service-account
    namespace: webapp-namespace
roleRef:
  kind: Role
  name: webapp-role
  apiGroup: rbac.authorization.k8s.io
EOF
```

# Deploying an application to our namepace

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp-deployment
  namespace: webapp-namespace
spec:
  selector:
    matchLabels:
      app: webapp
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: scottyc/webapp:latest
        ports:
        - containerPort: 3000
          hostPort: 3000
EOF
```

# Now let's set up kubectl to use the service account


@scottcoulton

# We need the secret for the service account

```
SECRET_NAME=$(kubectl get sa webapp-service-
account --namespace webapp-namespace -o json |
jq -r .secrets[].name)
```

# We will get the ca

```
kubectl get secret --namespace webapp-namespace
"${SECRET_NAME}" -o json | jq -r
'.data["ca.crt"]' | base64 --decode > ca.crt
```

# We will get the user token

```
kubectl get secret --namespace webapp-namespace
"${SECRET_NAME}" -o json | jq -r
'.data["ca.crt"]' | base64 --decode > ca.crt
```

# Then we will create our kubeconfig file

```
context=$(kubectl config current-context)

CLUSTER_NAME=$(kubectl config get-contexts "$context" | awk '{print $3}' | tail -n 1)

ENDPOINT=$(kubectl config view -o jsonpath="{.clusters[?(@.name == \"${CLUSTER_NAME}\")].cluster.server}")

kubectl config set-cluster "${CLUSTER_NAME}" --kubeconfig=admin.conf --server="${ENDPOINT}" --certificate-
authority=ca.crt --embed-certs=true

kubectl config set-credentials "webapp-service-account-webapp-namespace-${CLUSTER_NAME}" --
kubeconfig=admin.conf --token="${USER_TOKEN}"

kubectl config set-context "webapp-service-account-webapp-namespace-${CLUSTER_NAME}" --kubeconfig=admin.conf
--cluster="${CLUSTER_NAME}" --user="webapp-service-account-webapp-namespace-${CLUSTER_NAME}" --namespace
webapp-namespace

kubectl config use-context "webapp-service-account-webapp-namespace-${CLUSTER_NAME}" --
kubeconfig="${KUBECFG_FILE_NAME}"
```

# Export the file to use in the current terminal

```
export KUBECONFIG=admin.conf
```

# Use your kubectl commands to see what you have access too


@scottcoulton

# Statefull sets


@scottcoulton

# Stateful sets

Most applications will need some form of state. This is where stateful sets come in handy.  This will allow us to persist our data

Before we can access the volumes we need to set up

- storage class
- pvc ( persistent volume claim)

# Storage classes

Storage classes are the mechanism to provision storage on various backends.

Some of the common backends are

- Local disk, NFS, ISCSI
- Cloud disks ( Azure disk, AWS EBS)
- Advanced replicated storage ( Rook, Portworx)

# Storage classes Azure disk

In this course we will use the dynamic storage class that ships with AKS


@scottcoulton

# Persistent volume claims

Persistent volume claims are the units of storage that can be attached to pods.

PVC are configured into two classes

- static
- dynamic

# Creating a static claim

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: aks-volume-claim
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
EOF
```

# Using the claim

```
cat <<EOF | kubectl apply -f -
kind: Pod
apiVersion: v1
metadata:
  name: nginx-pvc
spec:
  volumes:
    - name: nginx-storage
      persistentVolumeClaim:
       claimName: aks-volume-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: nginx-storage
EOF
```

@scottcoulton

# Ingress controller

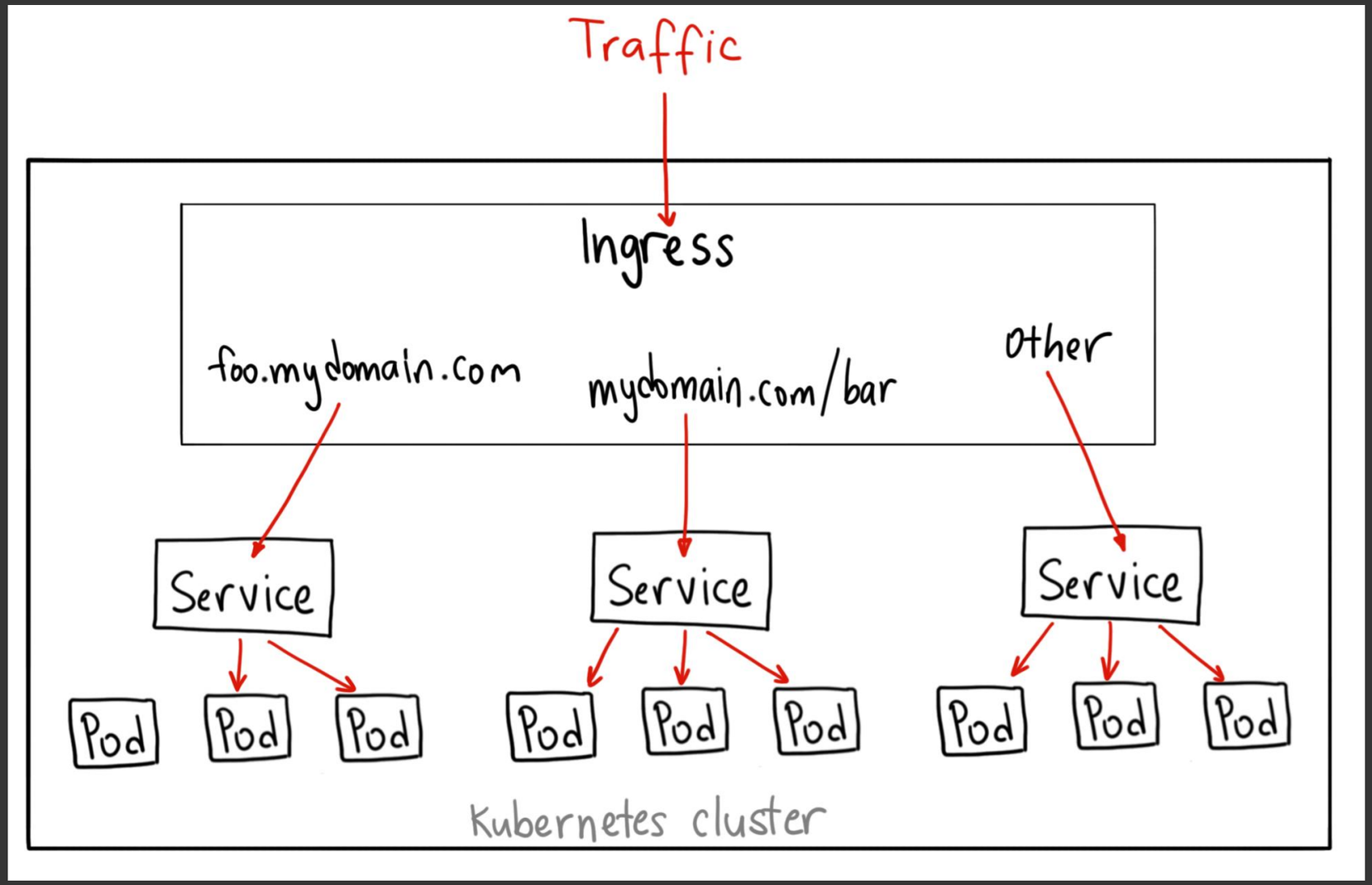

@scottcoulton

# Ingress controllers

We have services what do ingress controllers give us.

- Layer 7 routing
- ssl termination
- Single IP address for multiple services

# Ingress controllers



# Azure HTTP routing addon

Disclaimer !!! This addon is not for production use.

https://docs.microsoft.com/azure/aks/http-application-routing

@scottcoulton

# Azure HTTP routing addon

The addon gives us two things straight out of the box.

- A single ingress controller

The Ingress controller is exposed to the internet by using a Kubernetes service of type LoadBalancer. The Ingress controller watches and implements Kubernetes Ingress resources, which creates routes to application endpoints.

- External-DNS controller

Watches for Kubernetes Ingress resources and creates DNS A records in the cluster-specific DNS zone.

@scottcoulton

# Enable the addon

```
az aks enable-addons --resource-group k8s --
name k8s --addons http_application_routing
```

# Deploy our application

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: webapp-deployment
spec:
  selector:
    matchLabels:
      app: webapp
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: scottyc/webapp:latest
        ports:
        - containerPort: 3000
          hostPort: 3000
EOF
```

@scottcoulton

# Deploy a service

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Service
metadata:
  name: webapp
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 3000
  selector:
    app: webapp
  type: ClusterIP
EOF
```

# Add an ingress rule

```
DNS=$(az aks show --resource-group k8s --name k8s --query
addonProfiles.httpApplicationRouting.config.HTTPApplicationRoutingZoneName -o tsv)

cat <<EOF | kubectl apply -f -
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: webapp
  annotations:
    kubernetes.io/ingress.class: addon-http-application-routing
spec:
  rules:
  - host: webapp.$DNS
    http:
      paths:
      - backend:
          serviceName: webapp
          servicePort: 80
        path: /
EOF
```

# Virtual node with Virtual Kubelet

# What is virtual kubelet ?

Virtual Kubelet is an opensource implementation of kubelet. This allows kubernetes to schedule pods on platorms that have no nodes

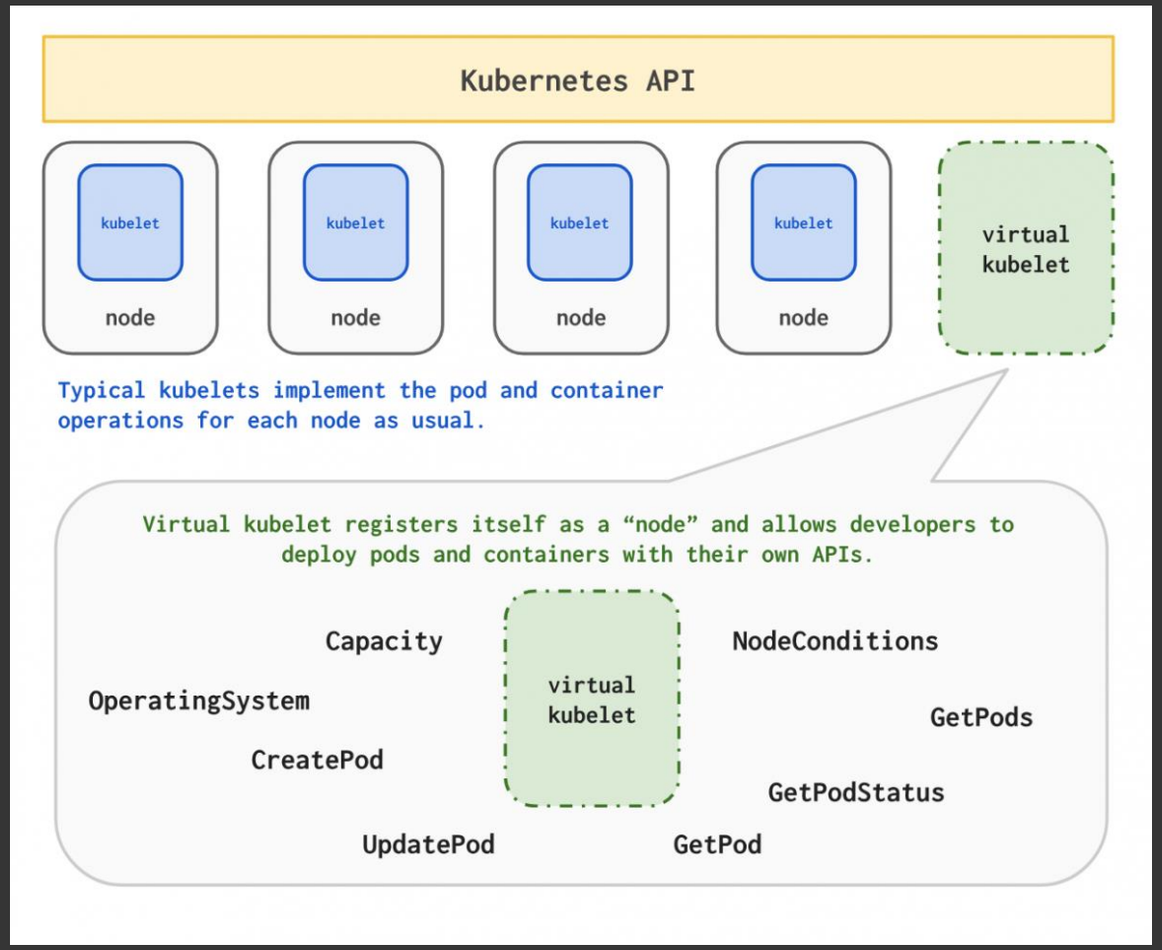
- Azure container service
- AWS Fargate
- Hyper.sh

# What does virtual kubelet give us

Virtual Kubelet gives us

- Burstable workloads
- Serverless intergration
- The ability to run Windows and Linux containers
- Cost savings

# Virtual kubelet




@scottcoulton

# Deploying Virtual Kubelet

```
az aks install-connector --resource-group k8s --name k8s --os-type both
```

# Test your Virtual Kubelet

```
kubectl get nodes
```

# Deploying a pod to virtual kubelet

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: vk-webapp
spec:
  containers:
  - image: scottyc/webapp
    imagePullPolicy: Always
    name: vk-webapp
    resources:
      requests:
        memory: 1G
        cpu: 1
    ports:
    - containerPort: 3000
      name: http
      protocol: TCP
  dnsPolicy: ClusterFirst
  nodeSelector:
    kubernetes.io/role: agent
    beta.kubernetes.io/os: linux
    type: virtual-kubelet
  tolerations:
  - key: virtual-kubelet.io/provider
    operator: Exists
  - key: azure.com/aci
    effect: NoSchedule
EOF
```

@scottcoulton

# Deploying an application on to virtual kubelet

```
nodeSelector:
    kubernetes.io/role: agent
    beta.kubernetes.io/os: linux
    type: virtual-kubelet
  tolerations:
  - key: virtual-kubelet.io/provider
    operator: Exists
  - key: azure.com/aci
    effect: NoSchedule
```

@scottcoulton

# Test your Virtual Kubelet

```
Kubectl get pods –o wide
```

Then use the public ip to access our application in the browser

**http://<public_ip>:3000**

# Pod security context

@scottcoulton

# Pod security context

Just because we are using Kubernetes means we are secure by defaut.

There are a lot of good security features in Kubernetes that are not turned on.

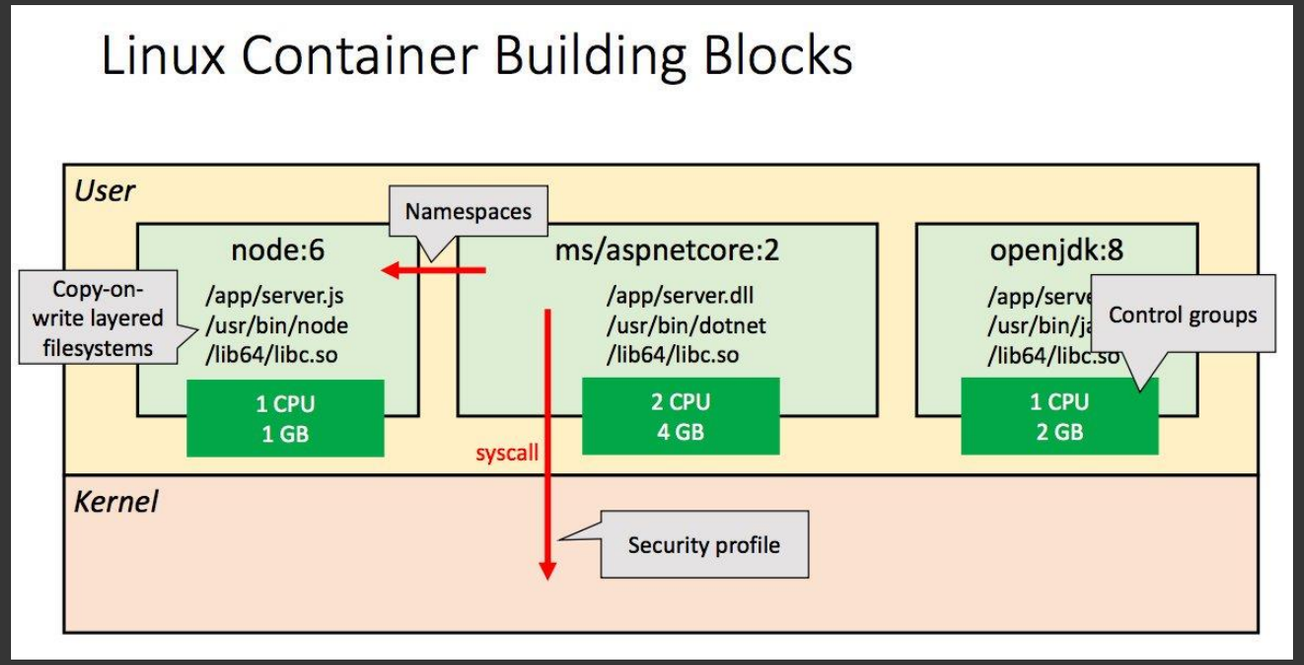
# Pod security context

Pod security is an abstraction from the Linux secuirty subsytem.

- Apparmor
- Selinux
- Secomp

# Pod security context

A container is a process that is isolated via kernel namespaces and cgroups




@scottcoulton

# Pod security context

## In Azure our pods are talking to the kernel via Moby

Today we are going to look at three of the important default policies.

In a production environment I would personally use secomp
https://www.kernel.org/doc/html/v4.16/userspace-api/seccomp_filter.html
With *libsecomp*
https://github.com/seccomp/libseccomp

@scottcoulton

# Pod security context

The three default policies are

- runAsUser
- readOnlyRootFilesystem
- allowPrivilegeEscalation

# Pod security context

We are going to run a series of deployments with our webapp changing the security context each time.

First, we must see why security context are so important.

# Deploy our webapp

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: webapp-deployment
spec:
  selector:
    matchLabels:
      app: webapp
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: scottyc/webapp:latest
        ports:
        - containerPort: 3000
          hostPort: 3000
EOF
```

@scottcoulton

# Check your deployment

`kubectl get svc` (to get our public ip)

Check your browser with `http://<public-ip>:3000`

# The hack

`kubectl get pods | grep webapp` (to get your pod name)

We will then get a shell inside our container
kubectl exec -it <pod_name> sh

Change the index.html file
`cd static && vim index.html`

Change the url on line 16
to `https://media.giphy.com/media/DBfYJqH5AokgM/giphy.gif`

# The hack

See what user is the pod running as

```
whoami from inside the pods terminal
```

# Check your deployment

`kubectl get svc` (to get our public ip)

Check your browser again with `http://<public-ip>:3000`

`Make sure you refresh your browser`

# Defining pod security context

Pod security policy is set in your deployment yaml under.

```
spec:
  containers:
    securityContext:
```

# Adding runAsUser policy

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: webapp-deployment
spec:
  selector:
    matchLabels:
      app: webapp
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: scottyc/webapp:latest
        ports:
        - containerPort: 3000
          hostPort: 3000
        securityContext:
          runAsUser: 1000
EOF
```

@scottcoulton

# Check the user running in the container

`kubectl get pods | grep webapp (to get your pod name)`

We will then get a shell inside our container

`kubectl exec -it <pod_name> sh`

Check the user with

`whoami`

Now let's test if we can change the file ? Or change to root

# Clean up

```
kubectl delete deployments.apps webapp-
deployment
```

# Adding readOnlyRootFilesystem policy

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: webapp-deployment
spec:
  selector:
    matchLabels:
      app: webapp
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: scottyc/webapp:latest
        ports:
        - containerPort: 3000
          hostPort: 3000
        securityContext:
          readOnlyRootFilesystem: true
EOF
```

@scottcoulton

# Check the readonly file system

`kubectl get pods | grep webapp (to get your pod name)`

We will then get a shell inside our container

`kubectl exec -it <pod_name> sh`

Now let's test if we can change the file ?

# Clean up

```
kubectl delete deployments.apps webapp-
deployment
```

# Adding allowPrivilegeEscalation policy

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: webapp-deployment
spec:
  selector:
    matchLabels:
      app: webapp
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: scottyc/webapp:latest
        ports:
        - containerPort: 3000
          hostPort: 3000
        securityContext:
          allowPrivilegeEscalation: false
EOF
```

@scottcoulton

# Check the readonly file system

`kubectl get pods | grep webapp (to get your pod name)`

We will then get a shell inside our container

`kubectl exec -it <pod_name> sh`

Now let's test if we can change the file ? Or change to root

# Clean up

```
kubectl delete deployments.apps webapp-
deployment
```

# Adding all three polices

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: webapp-deployment
spec:
  selector:
    matchLabels:
      app: webapp
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: scottyc/webapp:latest
        ports:
        - containerPort: 3000
          hostPort: 3000
        securityContext:
          runAsUser: 1000
          readOnlyRootFilesystem: true
          allowPrivilegeEscalation: false
EOF
```

@scottcoulton

# Check with all policies applied

Now let's run all our tests again on this pod

# Clean up

```
kubectl delete deployments.apps webapp-
deployment
```

# Introduction into istio


@scottcoulton

# Introduction into istio

## Istio gives us

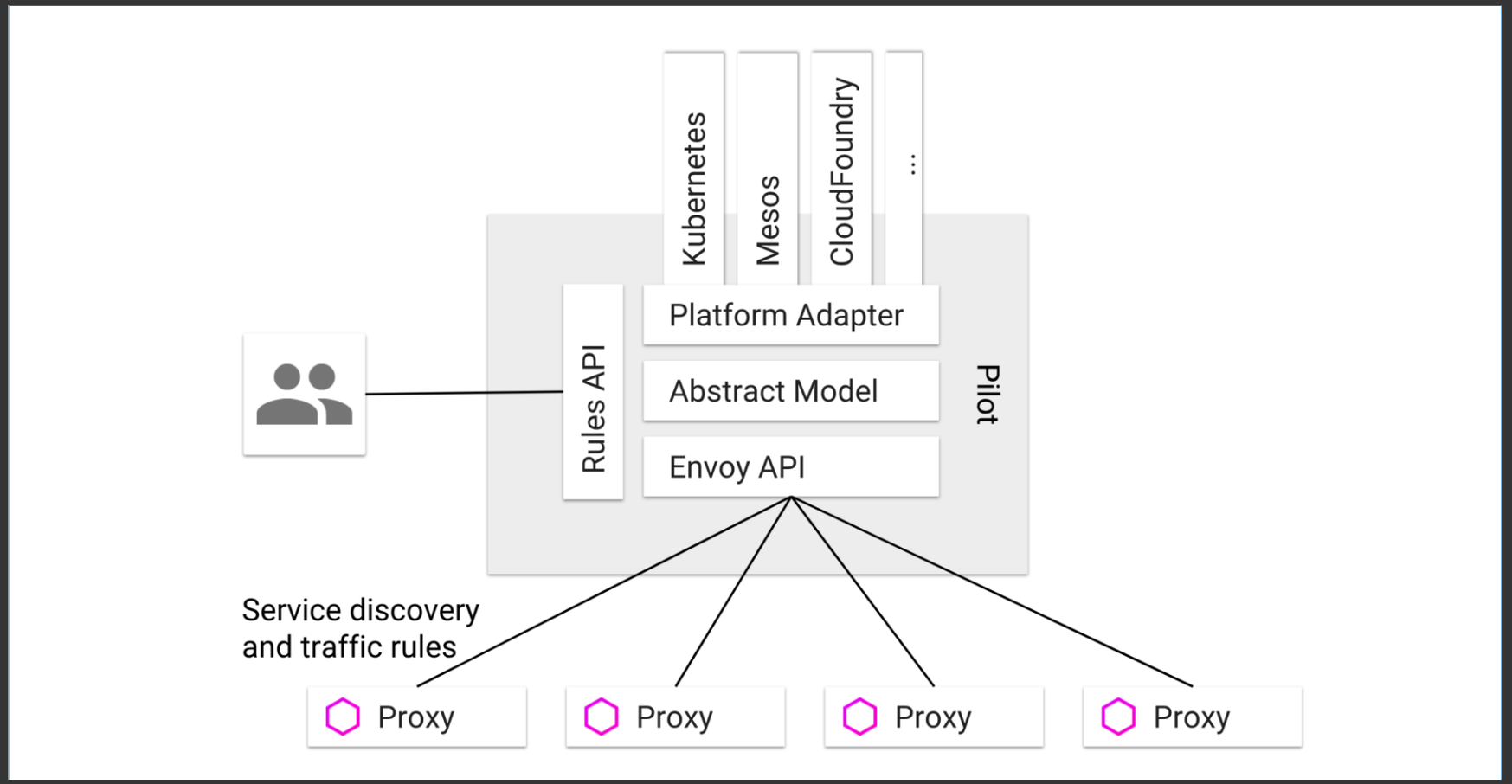
- Automatic load balancing for HTTP, gRPC, WebSocket, and TCP traffic.
- Fine-grained control of traffic behavior with rich routing rules, retries, failovers, and fault injection.
- A pluggable policy layer and configuration API supporting access controls, rate limits and quotas.
- Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress.
- Secure service-to-service communication in a cluster with strong identity-based authentication and authorization.

# Istio components

Istio is made up of the following components

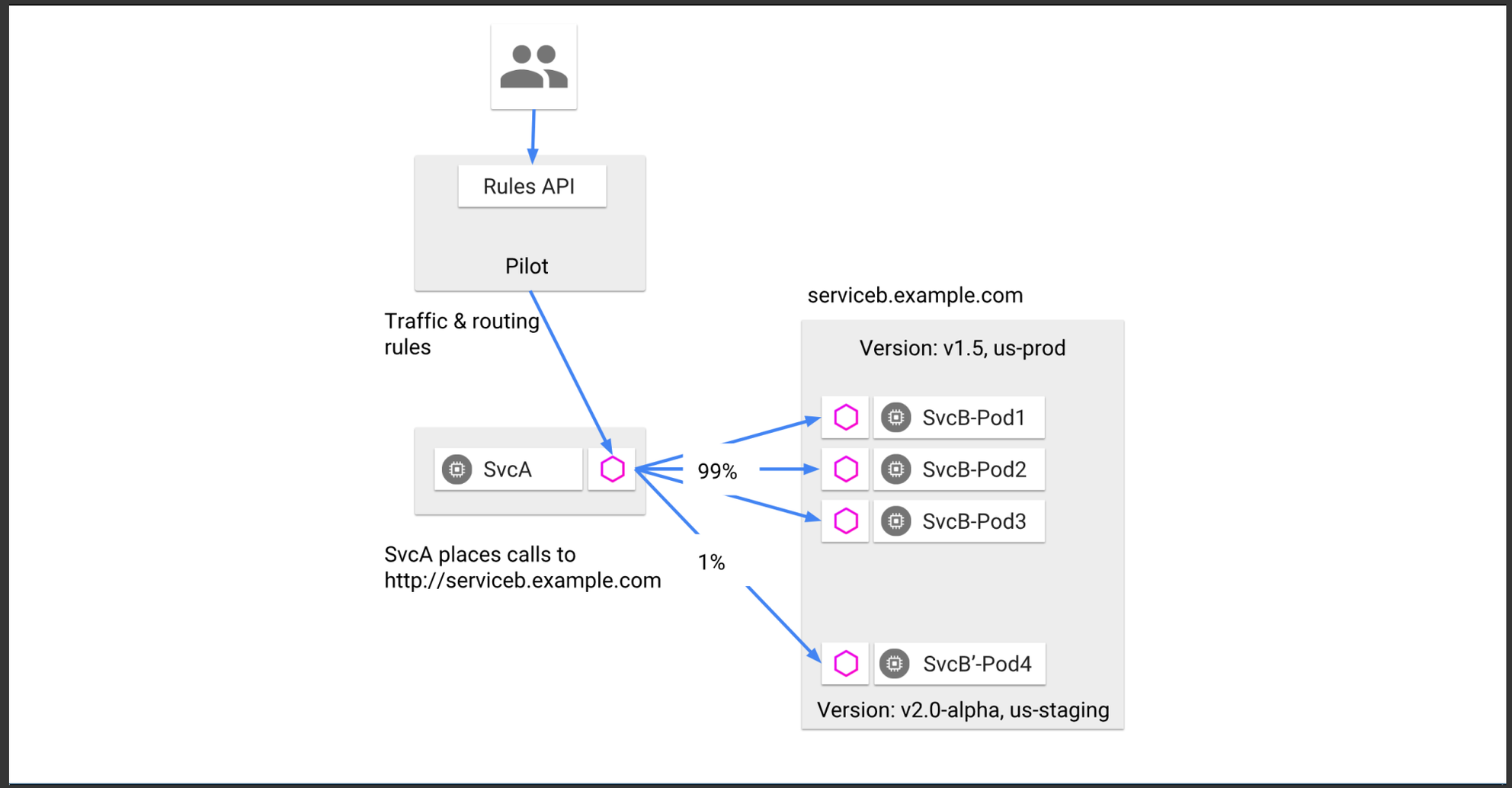
- Envoy and Piolit (ingress, egress & authentication policies)
- Citadel (certificate management)
- Mixer (authorization and auditing)

# Envoy and Piolit

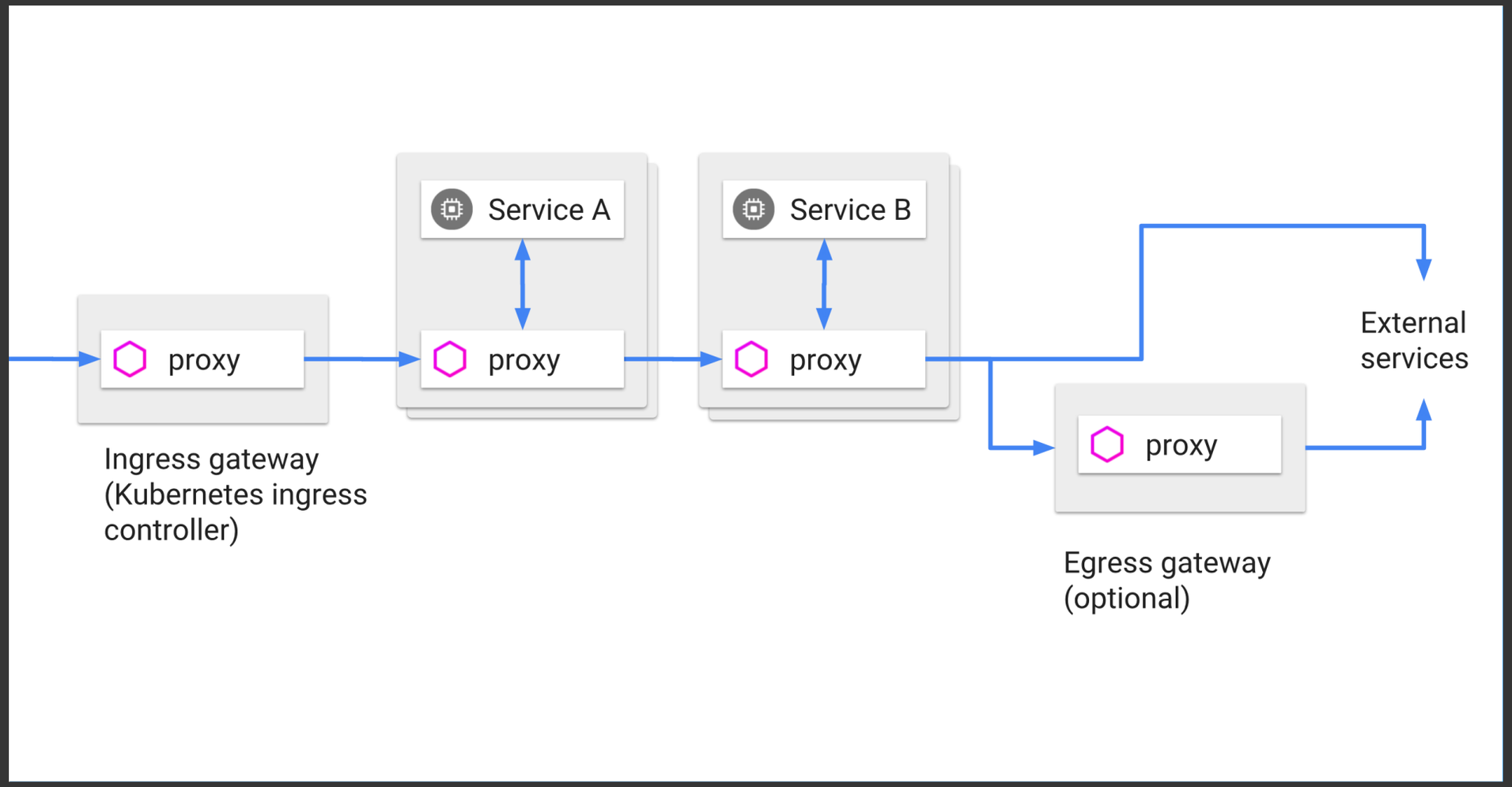



@scottcoulton

# Communication between services

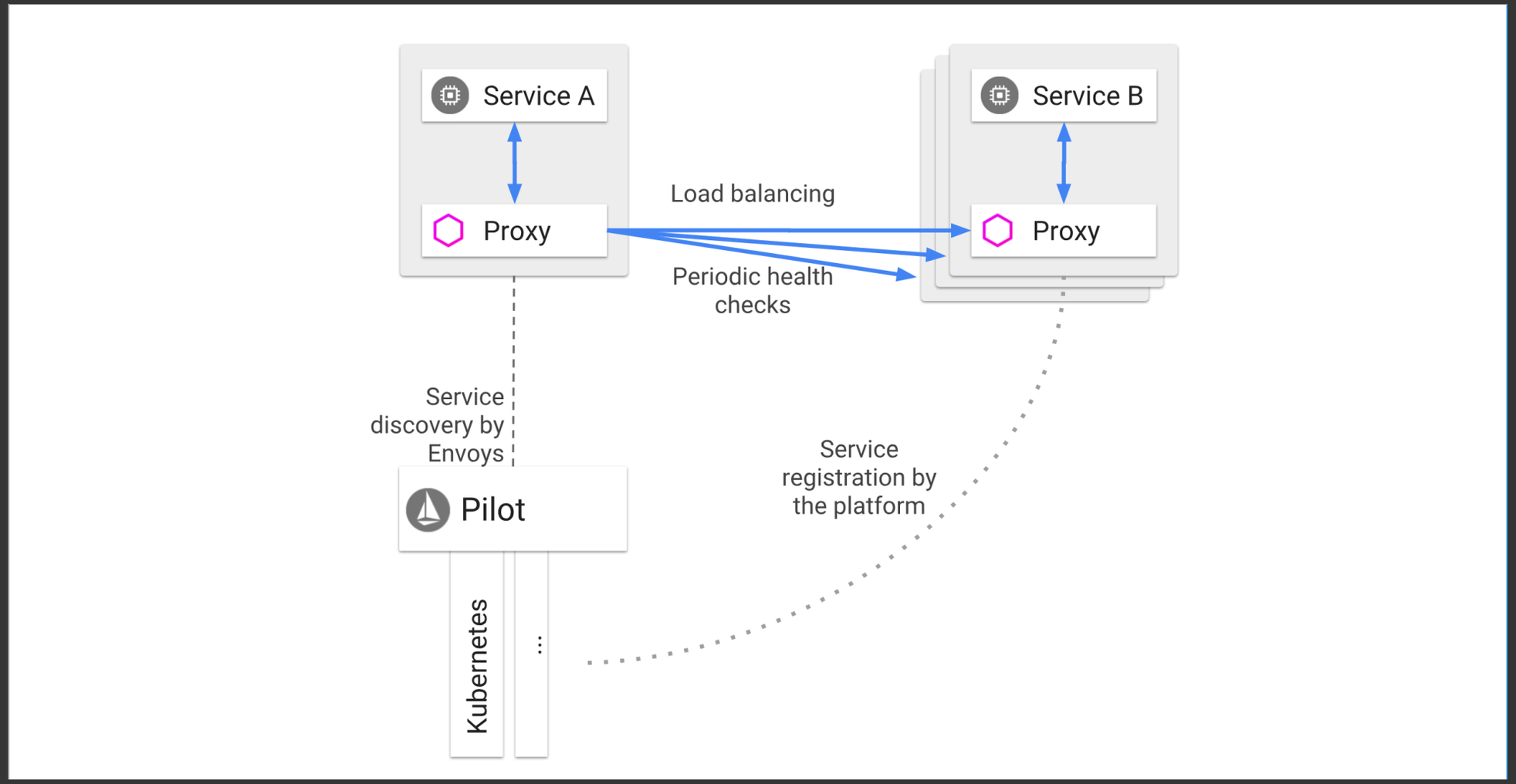


# Ingress and egress

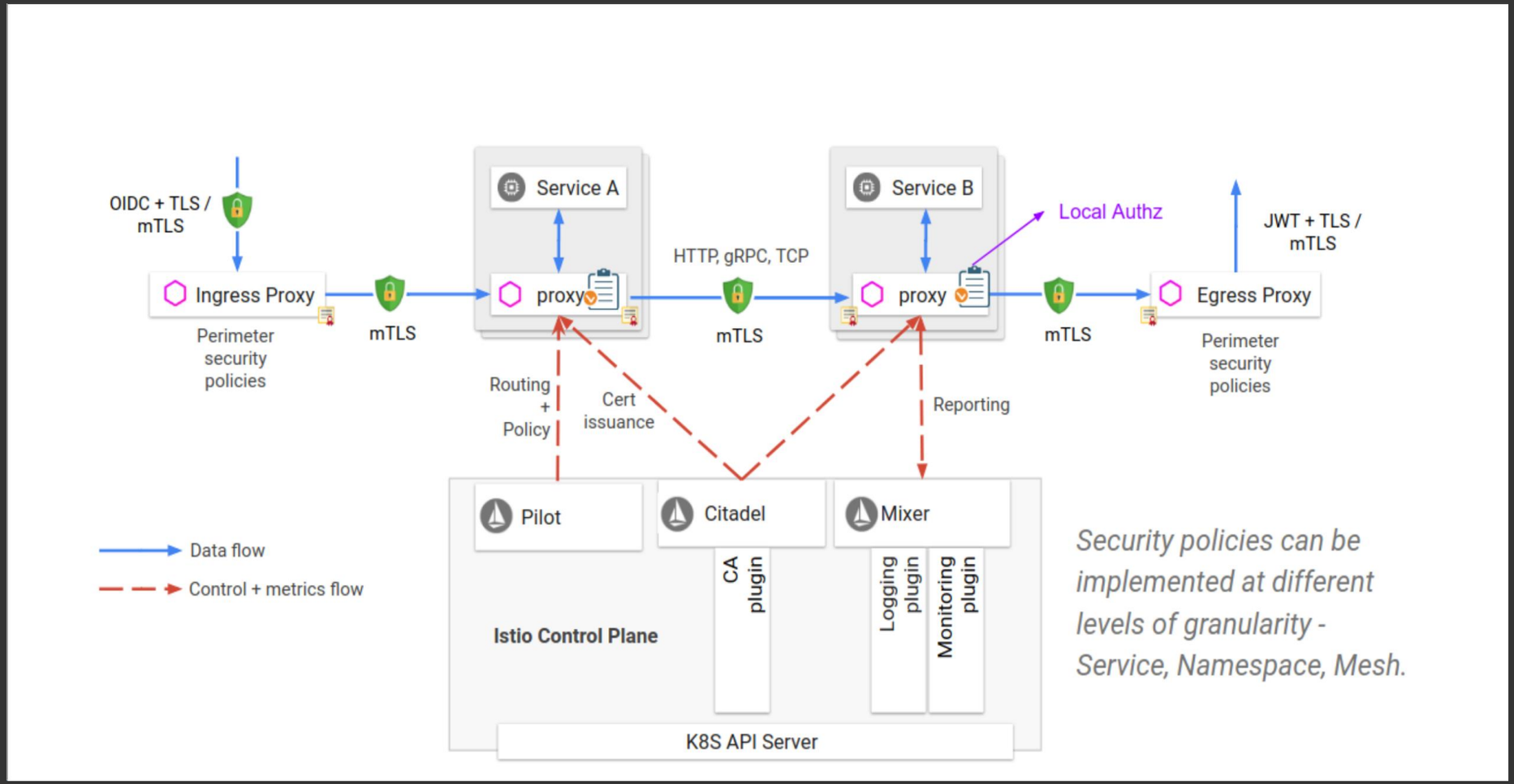



@scottcoulton

# Discovery and load balancing



# Certificate architecture




@scottcoulton

# Advanced application routing with istio


@scottcoulton

# Install istio from the script provided

Once istio is installed run

```
kubectl label namespace default istio-injection=enabled
```

# Istio application

We will use istio to deploy our webapp

- We will have two versions of the application v1 and v2
- We will weight the traffic evenly between the two
- Now we have the istio crd's installed we will use native kubernetes yamls files to deploy.

# Istio application architecture

Add diagram here !!!!

# Deploy the services that istio will talk to

```
apiVersion: v1
kind: Service
metadata:
  name: webapp
  labels:
    app: webapp
spec:
  ports:
  - port: 3000
    name: http
  selector:
    app: webapp
```

# Create two deployments of the web app

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: webapp-v1
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp
        version: v1
    spec:
      containers:
      - name: webapp
        image: scottyc/webapp:v1
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 3000
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: webapp-v2
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp
        version: v2
    spec:
      containers:
      - name: webapp
        image: scottyc/webapp:v2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 3000
```

# Create our destination rules

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: webapp
spec:
  host: webapp
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

# Create our gateway

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
name: webapp-gateway
spec:
selector:
istio: ingressgateway # use istio default controller
servers:
- port:
number: 80
name: http
protocol: HTTP
hosts:
- "*"
```

# Create our virtualservice

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
name: webapp
spec:
hosts:
- "*"
gateways:
- webapp-gateway
http:
- route:
- destination:
host: webapp
subset: v1
weight: 50
- destination:
host: webapp
subset: v2
weight: 50
```

# Get your ingress ip

```
kubectl get svc istio-ingressgateway -n istio-system -o
jsonpath="{.status.loadBalancer.ingress[0].ip}"
```

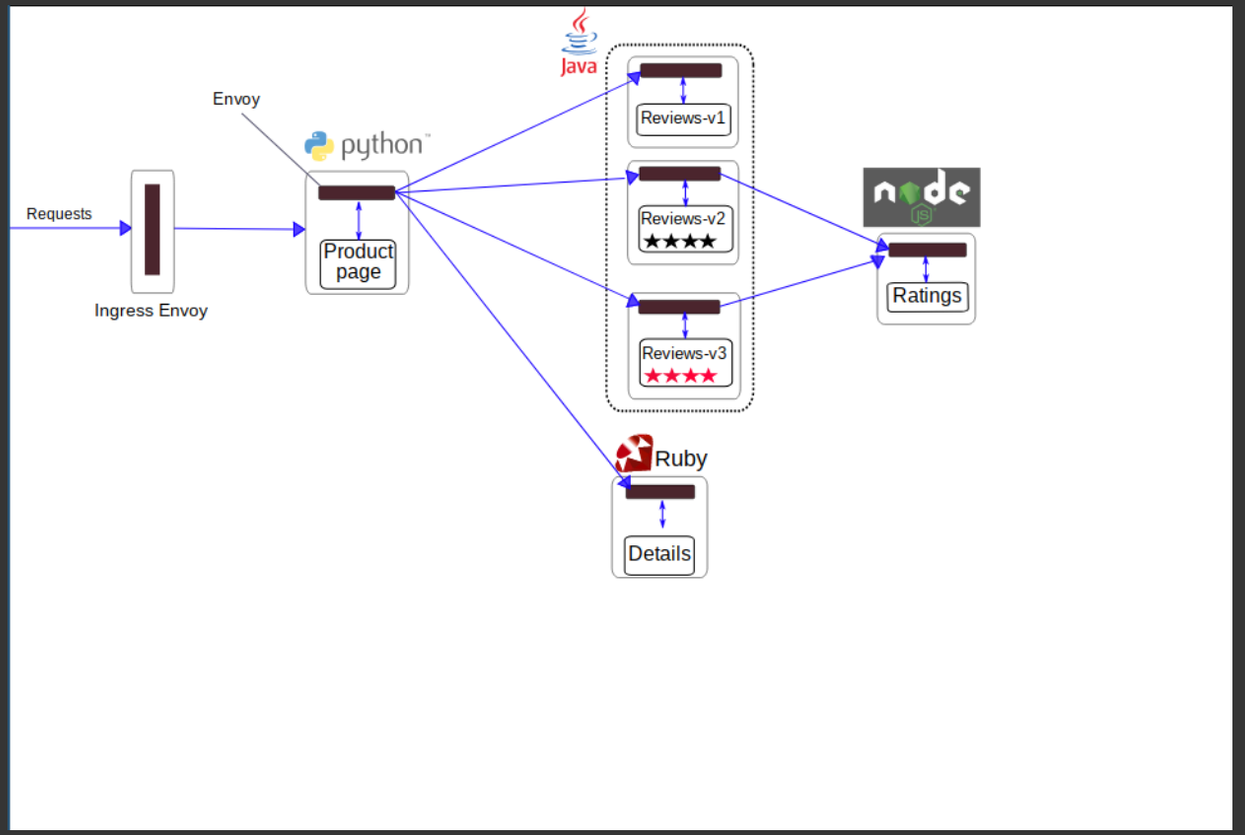
# mTLS with istio

@scottcoulton

# Warning !!!!!

mTLS needs to be set up in an empty namespace.

# The application we are deploying




@scottcoulton

# Create our namespace

```
kubectl create namespace istio-app
```

Make sure we turn on instio injection to our new namespace

```
kubectl label namespace istio-app istio-injection=enabled
```

# Enable mTLS across our namespace

```
cat <<EOF | istioctl create -f -
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: default
  namespace: istio-app
spec:
  peers:
  - mtls:
EOF
```

# Create a destination rule

```
cat <<EOF | istioctl create -f -
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: default
  namespace: istio-app
spec:
  host: "*"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
EOF
```

# Deploy our application

```
kubectl create -n istio-app -f istio-
1.0.4/samples/bookinfo/platform/kube/bookinfo
.yaml
```

# Create our virtual service

```
cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
  namespace: istio-app
spec:
  gateways:
  - bookinfo-gateway
  hosts:
  - '*'
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        exact: /login
    - uri:
        exact: /logout
    - uri:
        prefix: /api/v1/products
    route:
    - destination:
        host: productpage
        port:
          number: 9080
EOF
```

# Create our gateway

```
cat <<EOF | kubectl apply -f -
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
  namespace: istio-app
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
EOF
```

# Create our gateway

```
kubectl -n istio-system get service istio-ingressgateway -o
jsonpath='{.status.loadBalancer.ingress[0].ip}'
```

We will then use the ip address in our browser to make sure our site is working correctly.
http://<YOUR_IP>/productpage

# Check our mTLS status

```
istioctl authn tls-check | grep .istio-
app.svc.cluster.local
```

# Check our mTLS status

Exec into our envoy proxy sidecar

```
export POD_NAME=$(kubectl get pods --
namespace=istio-app | grep details | cut -d' '
-f1)
kubectl exec -n istio-app -it $POD_NAME -c
istio-proxy /bin/bash
```

# Hit a service internally

```
curl -k -v http://details:9080/details/0
```

# Let's capture the traffic with tcpdump

```
IP=$(ip addr show eth0 | grep "inet\b" | awk
'{print $2}' | cut -d/ -f1)
sudo tcpdump -vvv -A -i eth0 '((dst port 9080)
and (net $IP))'
```

# From another terminal

```
curl -o /dev/null -s -w "%{http_code}\n"
http://$(kubectl -n istio-system get service
istio-ingressgateway -o
jsonpath='{.status.loadBalancer.ingress[0].ip}'
)/productpage
```

# Output

```
^[22:47:36.978639 IP (tos 0x0, ttl 64, id 19003, offset 0, flags [DF], proto TCP (6), length 60)
    10.244.0.12.50662 > details-v1-6764bbc7f7-x7x99.9080: Flags [S], cksum 0x162b (incorrect -> 0xb758), seq 2995501799, win 29200, options [mss 1460,sackOK,TS val 1887650117 ecr
0,nop,wscale 7], length 0
E..<J;@.@...
...
..    ..#x.........r..+.........
p.AE........
22:47:36.978681 IP (tos 0x0, ttl 64, id 19004, offset 0, flags [DF], proto TCP (6), length 52)
    10.244.0.12.50662 > details-v1-6764bbc7f7-x7x99.9080: Flags [.], cksum 0x1623 (incorrect -> 0x3e8d), seq 2995501800, ack 2809488904, win 229, options [nop,nop,TS val 1887650117 ecr
2183432464], length 0
E..4J<@.@...
...
..    ..#x.....uf......#.....
p.AE.$..
22:47:36.978742 IP (tos 0x0, ttl 64, id 19005, offset 0, flags [DF], proto TCP (6), length 254)
    10.244.0.12.50662 > details-v1-6764bbc7f7-x7x99.9080: Flags [P.], cksum 0x16ed (incorrect -> 0xe838), seq 0:202, ack 1, win 229, options [nop,nop,TS val 1887650117 ecr 2183432464],
length 202
E...J=@.@...
...
..    ..#x.....uf............
p.AE.$..............A......}.Q..d.....................+.../...    ...../.,.0.
.....5...|........7.5..2outbound|9080||details.istio-app.svc.cluster.local.....#.................................istio.......
........
22:47:36.979801 IP (tos 0x0, ttl 64, id 19006, offset 0, flags [DF], proto TCP (6), length 52)
    10.244.0.12.50662 > details-v1-6764bbc7f7-x7x99.9080: Flags [.], cksum 0x1623 (incorrect -> 0x38ac), seq 202, ack 1280, win 251, options [nop,nop,TS val 1887650118 ecr 2183432465],
length 0
E..4J>@.@...
...
..    ..#x.....uk......#.....
p.AF.$..
22:47:36.981099 IP (tos 0x0, ttl 64, id 19007, offset 0, flags [DF], proto TCP (6), length 1245)
    10.244.0.12.50662 > details-v1-6764bbc7f7-x7x99.9080: Flags [P.], cksum 0x1acc (incorrect -> 0xba00), seq 202:1395, ack 1280, win 251, options [nop,nop,TS val 1887650120 ecr
2183432465], length 1193
E...J?@.@...
...
..    ..#x.....uk............
p.AH.$......:...6..3..00..,0.............->..._..).....0..    *.H........0.1.0...U.
..k8s.cluster.local0...190108215913Z..190408215913Z0.1    0...U.
..    ..#x.....up....7.#.....
p.AM.$..
```

@scottcoulton

Microsoft

Azure ❤