SHORT

ISTIO DEMO

LINKERD VS ENVOY VS ISTIO

- Linkerd
  - https://linkerd.io/
  - Can be deployed as sidecar (per pod) or per host
- Envoy
  - https://www.envoyproxy.io/
- Most of the actual functionality is implemented by either of these services
- Envoy is lighter-weight from a memory/CPU standpoint
- Google is backing Envoy. Envoy is the default Istio backend.

RELATIONSHIP BETWEEN ISTIO AND ENVOY/LINKERD

- Istio (https://istio.io) provides the uniform API and K8S controllers
- Envoy/Linkerd implement the functionality
- Istio configures Envoy/Linkerd based on the configuration set in Istio

COMPONENTS

- **Envoy**
  Sidecar proxy intercepts communications and integrates with service mesh
- **Mixer**
  Collects telemetry data, enforces access control and policies, abstracts backends (such as K8S)
- **Pilot**
  Provides service discovery, traffic management, and resiliency. Abstracts backends (such as K8S)
- **Istio-Auth**
  Provides service-to-service and end-user authentication using TLS.

FEATURES

REQUEST ROUTING

- Understands concept of service versions
- Supports egress services (routing to external services)
- Route by source or header
- Split traffic between service versions
- Migrate to new service versions (traffic shifting)
- Service discovery
- Load balancing (including zone aware)
- Mirroring
- Rate Limiting

FAILURE HANDLING

- Request timeouts
  - Overridable via request headers
- Request retries
  - Bounded with timeout budget and variable jitter
- Circuit breakers
  - More of a load balancer ejection policy…

FAULT INJECTION

- Inject delays
- Inject aborts
- Can specify a percentage of requests that receive error
- Can control via same matching criteria as traffic routing

MONITORING, LOGGING AND TRACING

- Distributed Tracing (Zipkin or Jaeger)
- Collecting Metrics and Logs
  - Prometheus
  - Grafana
  - Fluentd

SECURITY

- Mutual TLS Authentication
- RBAC Service Authorization
  - Namespace level
  - Service level
  - Method level
  - Service-to-Service and User-to-Service

DEMO

SETUP

- Install Edge version of Docker for Mac
- Activate Kubernetes
  - *Preferences > Kubernetes > Enable Kubernetes*
- Switch kubectl context to docker-for-desktop
  - *Docker menu > Kubernetes > docker-for-desktop*
- Make sure you can do a **kubectl get namespace** from the command line
- Download istio: **curl -L https://git.io/getLatestIstio | sh -**
- Add istio bin to path: **export PATH="$PATH:$PWD/istio-0.7.1/bin"**

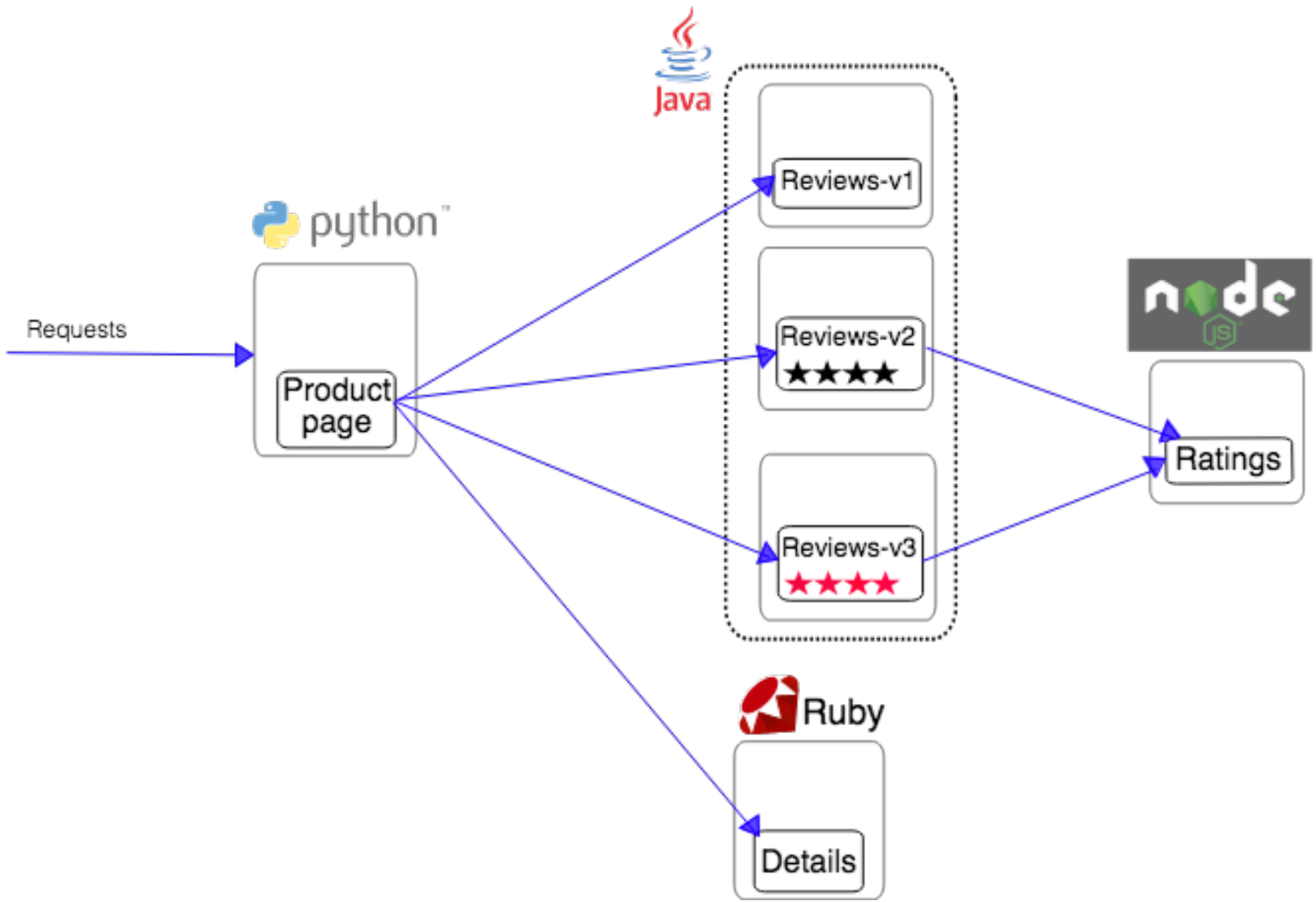
INSTALL ISTIO

- Install istio to K8S
  - **kubectl apply -f install/kubernetes/istio.yaml**
- Confirm running
  - **kubectl get svc -n istio-system**
  - **kubectl get pods -n istio-system**

INSTALL AUTOMATIC PROXY INJECTION

- Create certificates
  - **./install/kubernetes/webhook-create-signed-cert.sh --service istio-sidecar-injector --namespace istio-system --secret sidecar-injector-certs**
- Install ConfigMap
  - **kubectl apply -f install/kubernetes/istio-sidecar-injector-configmap-release.yaml**
- Configure cert into webhook resource
  - **cat install/kubernetes/istio-sidecar-injector.yaml | ./install/kubernetes/webhook-patch-ca-bundle.sh > install/kubernetes/istio-sidecar-injector-with-ca-bundle.yaml**
- Install injector webhook
  - **kubectl apply -f install/kubernetes/istio-sidecar-injector-with-ca-bundle.yaml**
- Enable automatic proxy injection
  - **kubectl label namespace default istio-injection=enabled**
- Confirm running
  - **kubectl -n istio-system get deployment -listio=sidecar-injector**

BOOKINFO SAMPLE APP



INSTALL BOOKINFO SAMPLE APP

- Install
  - **kubectl apply -f samples/bookinfo/kube/bookinfo.yaml**
- Check running
  - **kubectl get services**
  - **kubectl get pods**
  - **kubectl get ingress -o wide**
- Go to web page
  - **http://localhost/productpage**
  - Refresh a couple of times. You will see different ratings behaviors because no default route has been set for the reviews service.

SERVICE ROUTING EXAMPLE

- Look at *samples/bookinfo/kube/route-rule-all-v1.yaml*
- **istioctl create -f samples/bookinfo/kube/route-rule-all-v1.yaml**
  - istioctl vs kubectl
- Refresh page a few times. Should get V1 (no stars)
- Look at *samples/bookinfo/kube/route-rule-reviews-test-v2.yaml*
- **istioctl create -f samples/bookinfo/kube/route-rule-reviews-test-v2.yaml**
- Refresh page. Should get no stars.
- Log in as jason/jason
- Should get V2 (black stars version)

MONITORING

- Install Prometheus
  - **kubectl apply -f install/kubernetes/addons/prometheus.yaml**
- Set up proxy to dashboard
  - **kubectl -n istio-system port-forward $(kubectl -n istio-system get pod -l app=prometheus -o jsonpath='{.items[0].metadata.name}') 9090:9090 &**
- Send some traffic to the site
- Open view at http://localhost:9090

MONITORING

- Install Grafana
  - **kubectl apply -f install/kubernetes/addons/grafana.yaml**
- Set up proxy to dashboard
  - **kubectl -n istio-system port-forward $(kubectl -n istio-system get pod -l app=grafana -o jsonpath='{.items[0].metadata.name}') 3000:3000 &**
- Send some traffic to the site
- Open view at http://localhost:3000

DISTRIBUTED TRACING

- Install Jaeger
  - **kubectl apply -n istio-system -f https://raw.githubusercontent.com/jaegertracing/jaeger-kubernetes/master/all-in-one/jaeger-all-in-one-template.yml**
- Set up proxy to dashboard
  - **kubectl port-forward -n istio-system $(kubectl get pod -n istio-system -l app=jaeger -o jsonpath='{.items[0].metadata.name}') 16686:16686 &**
- Send some traffic to the site
- Open view at http://localhost:16686

FAULT INJECTION

- **git clone git@github.com:abuchananTW/istio-demo.git**
- **istioctl create -f istio-demo/fault-injection.yaml**
- Refresh page a number of times. You should see some where the details cannot be retrieved.
- Look at the Jaeger interface and inspect the traces to see the errors.

RETRIES

- **kubectl create -f istio-demo/fortio.yaml**
- **while true; do curl -o /dev/null -s -w "%{http_code}\n" "http://localhost?status=503:50"; done** for a bit and note that around half the requests fail.
- Look at the Jaeger traces to see that the request is only made once.
- **istioctl create -f istio-demo/fortio-retries.yaml**
- **while true; do curl -o /dev/null -s -w "%{http_code}\n" "http://localhost?status=503:50"; done** again, and note that most requests succeed.
- Look at the Jaeger traces to see that the requests is made up to 3 times.