

Experience Report: Multi-Cloud Serverless on Knative

Mark Wang

Evan Anderson

S&P Global Ratings Technology

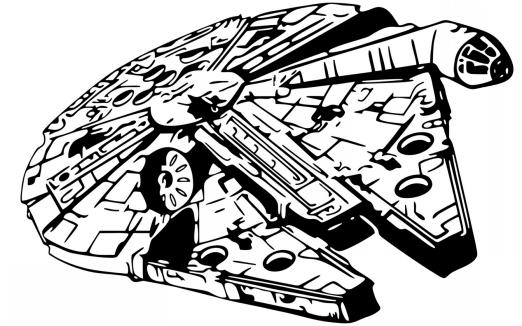
FaaS Overview:



Why FaaS?

Ship Faster

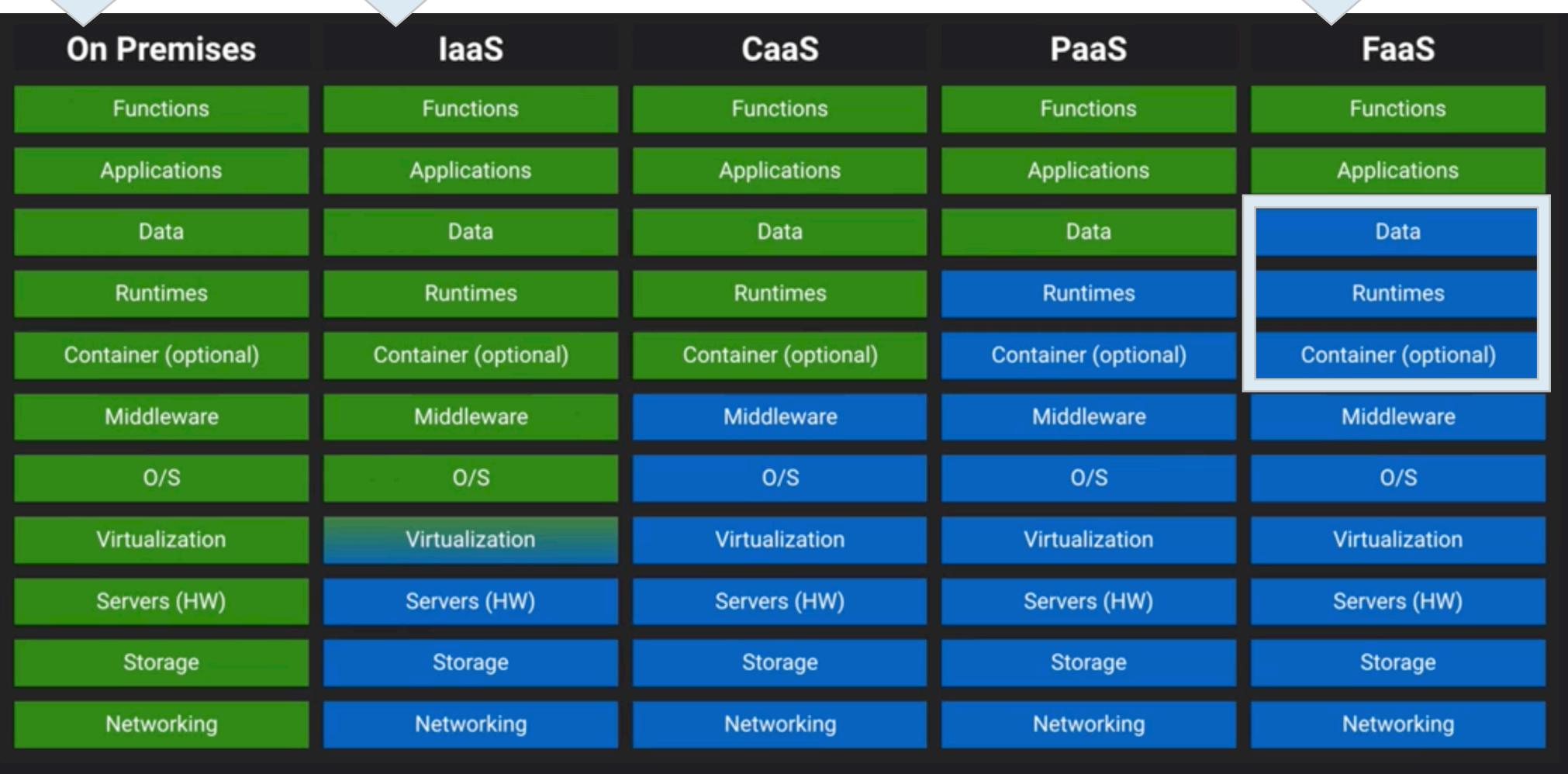
- Scrum teams will be able to **release frequently**
- Scrum teams will start to **own** more of the technology stack
- There will be **less tech debt**



Transform the Culture

- Partner with application developers in defining the platform
- Allow teams to unblock themselves on platform features

2020 Strategic Initiative: 100% Bronze FaaS Certification



FaaS Certification Roadmap

Phase	Dec 2019	Jan 2020	Q1	Q2	Q3	Q4		
Learn & Experiment	Technology Evaluation	Experiment Knative		Upgrade Knative	Additional Integrations	Additional Regions		
	Pilot Use Cases				Open Standards	FaaS Pipeline		
Build Foundation FaaS Accelerator()	Work Streams defined		FaaS MVP 1.0	FaaS 2.0	FaaS 3.0	FaaS 4.0		
	Certification & Training Program defined		Define Target states: FaaS, Containers, Retire		24x7 Support Setup	CaaS MVP 1.0		
FaaS Adoption	Pilot Apps Workshops & Trainings		Workshops & Trainings	Workshops & Trainings	Workshops & Trainings	Workshops & Trainings		
			Bronze Certification		Bronze Certification	Bronze Certification		
Application Migration	Select Reference App Patterns & Wave 1 Apps		FaaS Reference Implementations	Containers Reference Implementations				
	Adopt Open Contribution Model		Wave 1 Apps on FaaS	Wave 2 Apps on FaaS	Wave 3 Apps on FaaS			
Application Roadmap			Cross Functional Teams on FaaS/CaaS	Wave 1 Apps on Containers	Wave 2 Apps on Containers	Retire Use Cases		
			Retire Use Cases	Retire Use Cases	Retire Use Cases	Retire Use Cases		

Platform Release stages (released monthly)

MVP 1.x (Q1)

- Working base platform
 - Kubernetes: EKS
 - Knative
 - Istio
 - Azure DevOps
 - Base images
- Pilot applications only
- Define model and working processes
 - Open contribution model
- Initial training materials and documentation

GA 2.x (Q2)

- 24x7 support
- Additional platform capabilities
 - OIDC/OAuth
 - Logging & Monitoring
 - Onboarding automation
 - CI/CD pipeline automation
- Open for early adopters
- Reference implementations: Java, .Net, Python, Angular
- Improved training materials and documentation

GA 3.x (Q3)

- Recommended for all developers
- Additional platform capabilities
 - DMZ Support
 - Eventing
 - Distributed Tracing
 - Spot instances
- Add Container-as-a-Service options
- Additional integrations
 - Container Security
- Reference implementations: Static Web Hosting

FaaS 3.1 Feature Highlights

- Languages:



python™



- Eventing:



- Security:



- Logging & Monitoring:



Prometheus



- Distributed tracing:



- Authentication:



- Network: Internal and DMZ

- Spot instances

- Blue/Green cluster automation

FaaS Platform Components

Major Components in order of operation

1. AWS EKS

Managed Control Plane for Kubernetes

2. Kubernetes

Container Orchestration

3. Istio

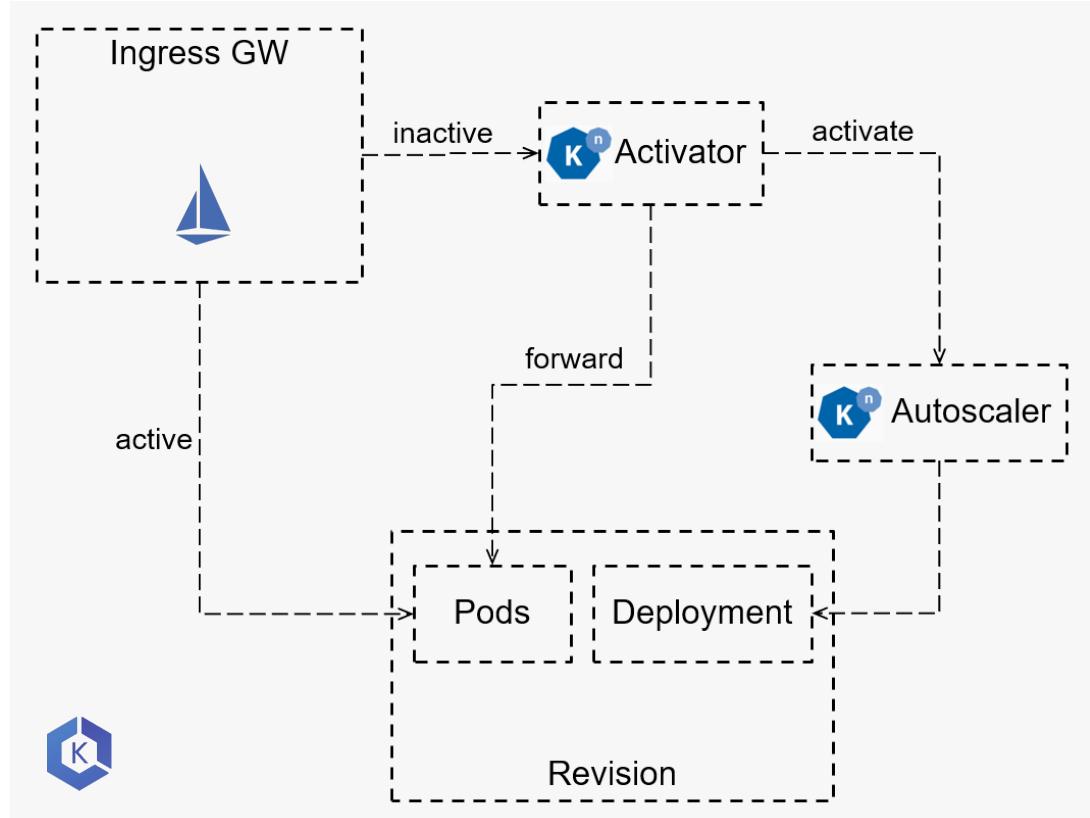
Ingress Control and Request Queuing

4. Knative

On-demand pod scaling and request routing, scale to zero

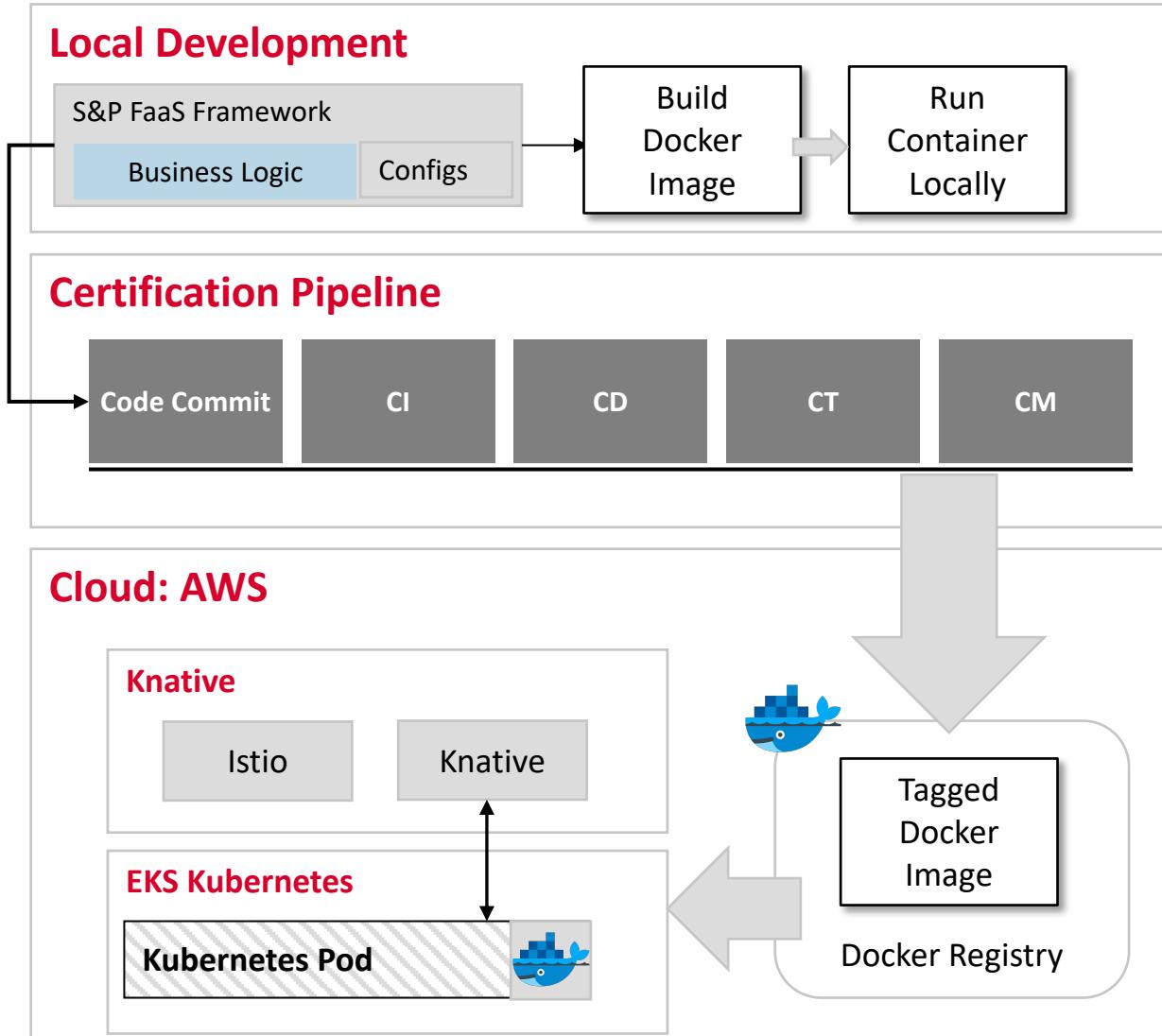
Single Knative Cluster consists of:

- 40+ AWS components
- Tens of thousands lines of YAML



Source: <https://istio.io/blog/2019/knative-activator-adapter/knative-activator.png>

FaaS Accelerator() (Knative)



Knative:

Provides FaaS capabilities on top of Kubernetes. It is a framework/set of extensions focusing on high-level abstractions for common application use cases like serving and eventing.

Local Development function code can be tested by running the Docker image locally in a container, as long as Knative-specific integrations are not used. In certain cases, deployment to a dev K8s cluster will be needed.

Certification Pipeline will deploy Business Logic (Function Code) as a Container Image to **Knative** and eventually land on Kubernetes such as AWS EKS

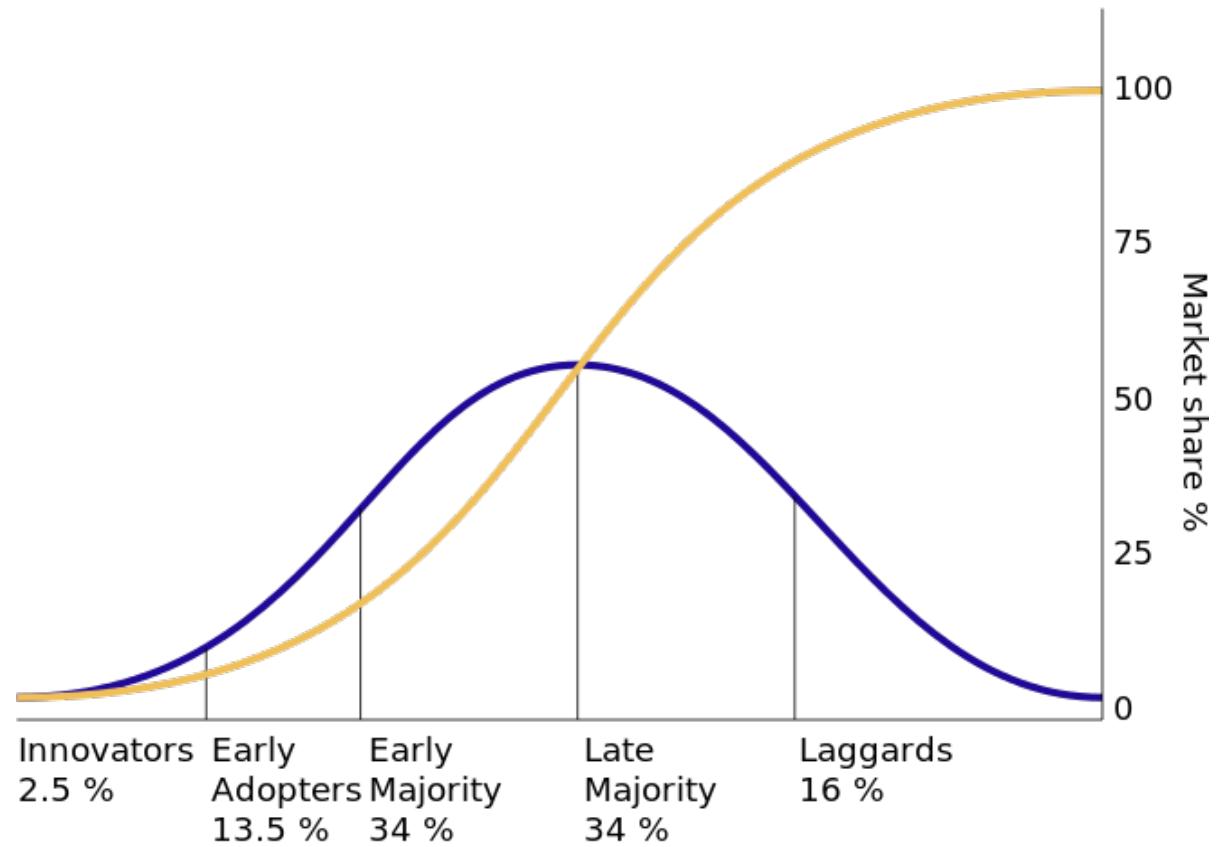
This is very portable; higher environments will simply deploy the gold copy of the Function artifact as Container Image

Demo

Adoption

In less than 1 year:

- 90% of the applications in scope have started or completed FaaS functions
- 50% of teams have written FaaS functions



What's this Knative stuff about?



Serverless *HTTP* applications

- Knative works with applications which expose an HTTP server on a known port.
- Why HTTP:
 - Ubiquitous, well-understood
 - Ongoing improvements with HTTP/2, Websockets, etc
 - Shares resources (IP, cert, routers) well
 - If we know the protocol, we can do :magic:



Kubernetes supports applications which use multiple protocols and ports, and which have many scaling models. The tradeoff is that Kubernetes requires a lot of settings even for a basic deployment.

Knative specializes Kubernetes for stateless HTTP applications. It's a complement to the core Kubernetes Deployment, StatefulSet, and Job abstractions.

Serverless HTTP Applications

```
apiVersion: apps/v1      apiVersion: v1
kind: Deployment          kind: Service
metadata:
  name: myapp
spec:
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - image: docker.io/test/myapp
      ports:
        - name: http
      containerPort: 8080
```

... less YAML, too



```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: myapp
spec:
  template:
    spec:
      containers:
        - image: docker.io/test/myapp
      ports:
        - name: http
```

This also includes an autoscaler and some other magic we'll talk about soon!

Serverless *HTTP* Applications

What else do you get out of the box with Knative?

- Autoscaling (already covered)
- Automatic management of HTTP hostnames
 - Map a wildcard address to your cluster, and no further need to touch DNS!
- Automatic tracking of previous states (Revisions)
 - Every update to the Service's spec creates a new Revision
 - These Revisions are immutable and automatically garbage collected
- Ability to roll back and canary traffic between Revisions
 - Can route traffic on a percentage basis, regardless of how many instances
 - A traffic route can refer to a specific Revision, or the last Ready Revision
 - Can also add tags to reach a specific Revision by a unique hostname – handy for testing

I promised some magic

Serverless HTTPS applications

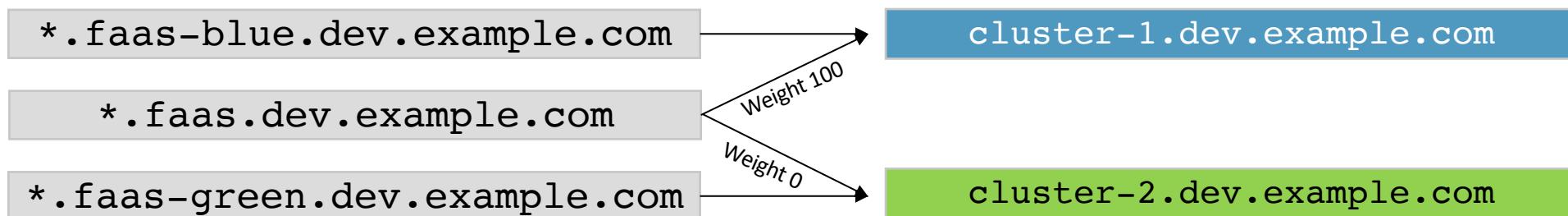
- Knative Serving can be configured to automatically provision certificates for your automatically-created hostnames.
- Out of the box, this uses LetsEncrypt and certmanager
- You could also integrate this with your own CA
- Supports both DNS (wildcard) and HTTP certificate solvers

Requirements:

- Your serving domain must be reachable by LetsEncrypt (i.e. on the internet)
- LetsEncrypt supports about 50 certs/week/domain prefix

SSL in the real world at S&P

- Cluster-specific DNS zones: *.`faas-blue`.`dev.example.com`
 - Contains a wildcard record pointing at the specific cluster
- Environment-specific DNS zones: *.`faas.dev.example.com`
 - Contains a wildcard record which points at the currently-active cluster (blue or green)
 - Can use DNS to perform active/active, active/passive, or weighted traffic routing
- AWS Certificate Manager SSL certificates provisioned for each cluster
 - AWS CA provisions SSL certificates covering both the cluster-specific and environment-specific names
 - Certificates are automatically created during cluster build with DNS validation for cluster SANs using AWS Route53
 - Certificates are installed on the ingress loadbalancers



Life of a Query

Knative Serving, part 1

Data Path

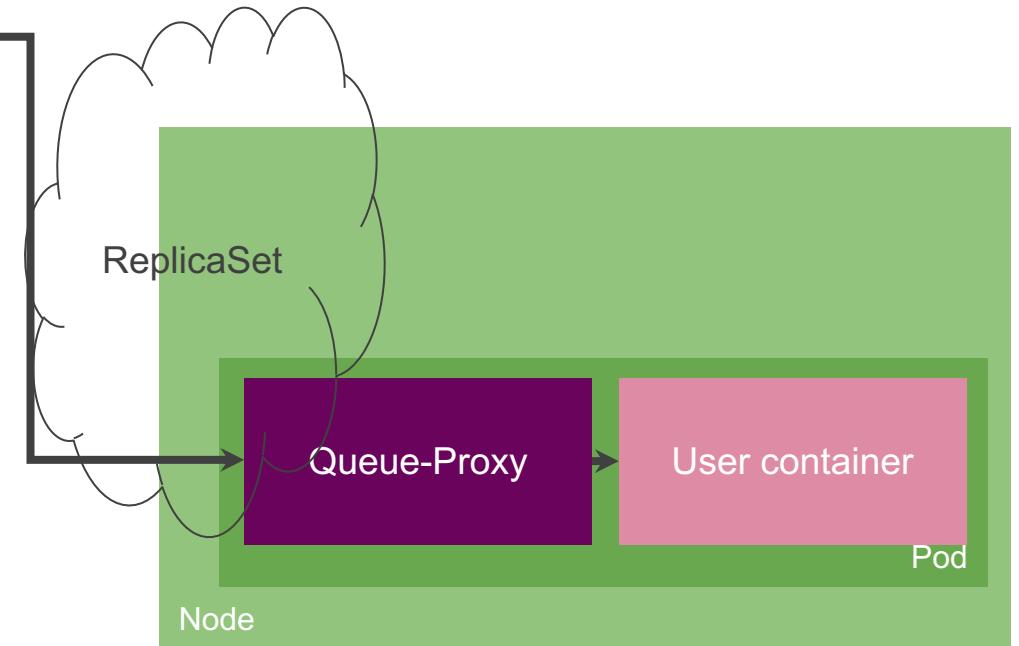
Life of a query

GET / HTTP/1.1

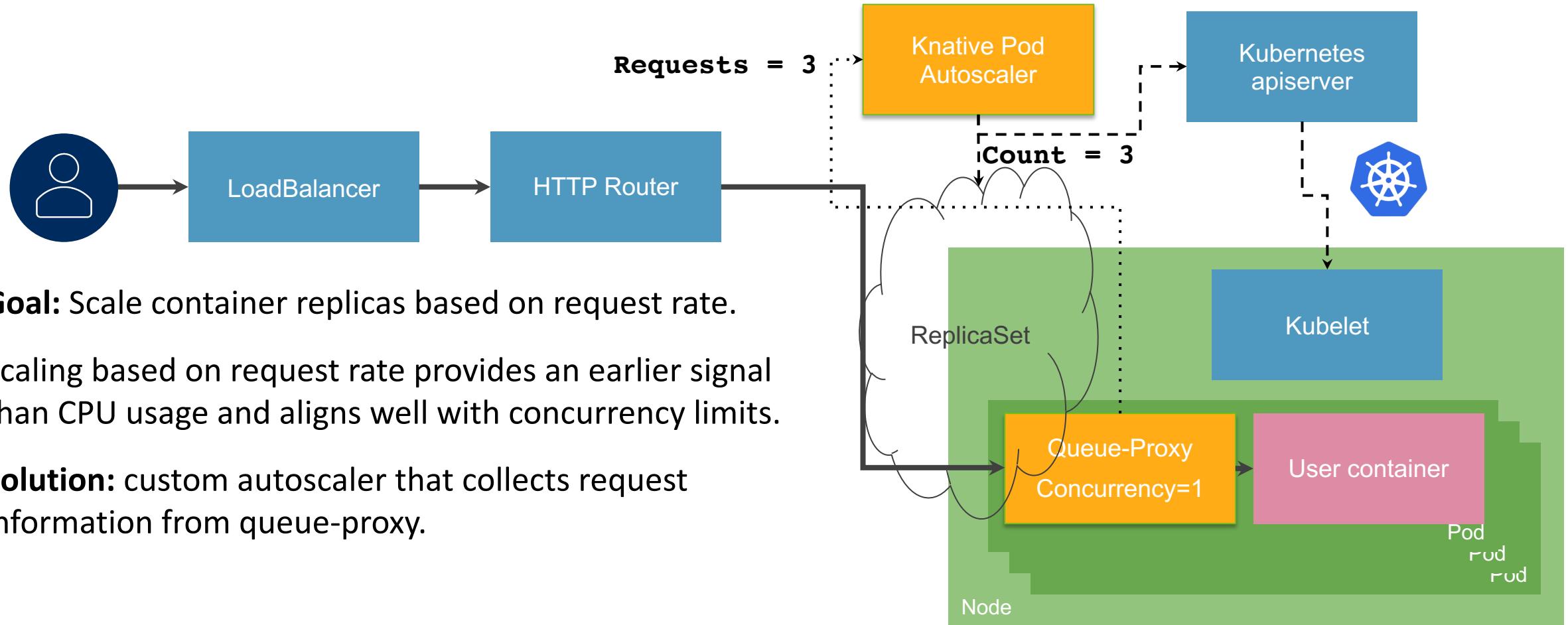


Goal: Steady-state should not be substantially more expensive than using “raw” Kubernetes.

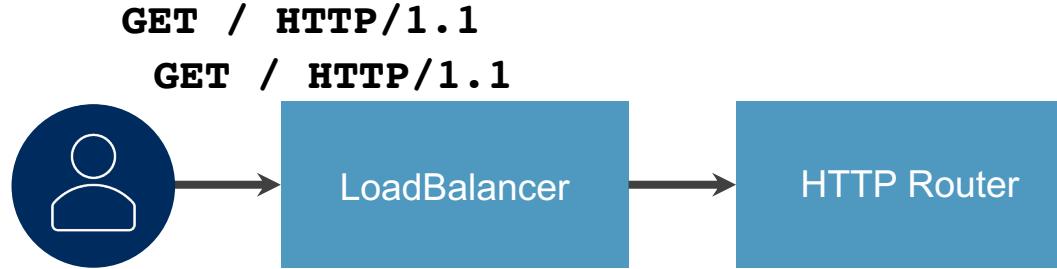
In particular, any middle-boxes that we add need to autoscale based on traffic demand. (And we prefer to have fewer such middleboxes, and to have them co-located with existing components in some way to reduce traffic via eth0).



Life of a query



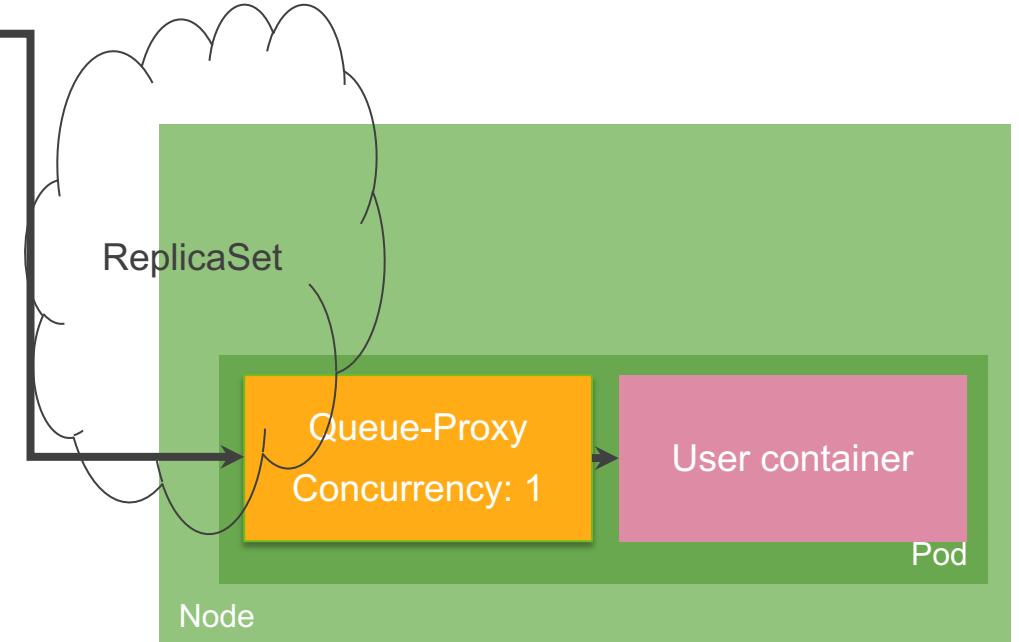
Life of a query



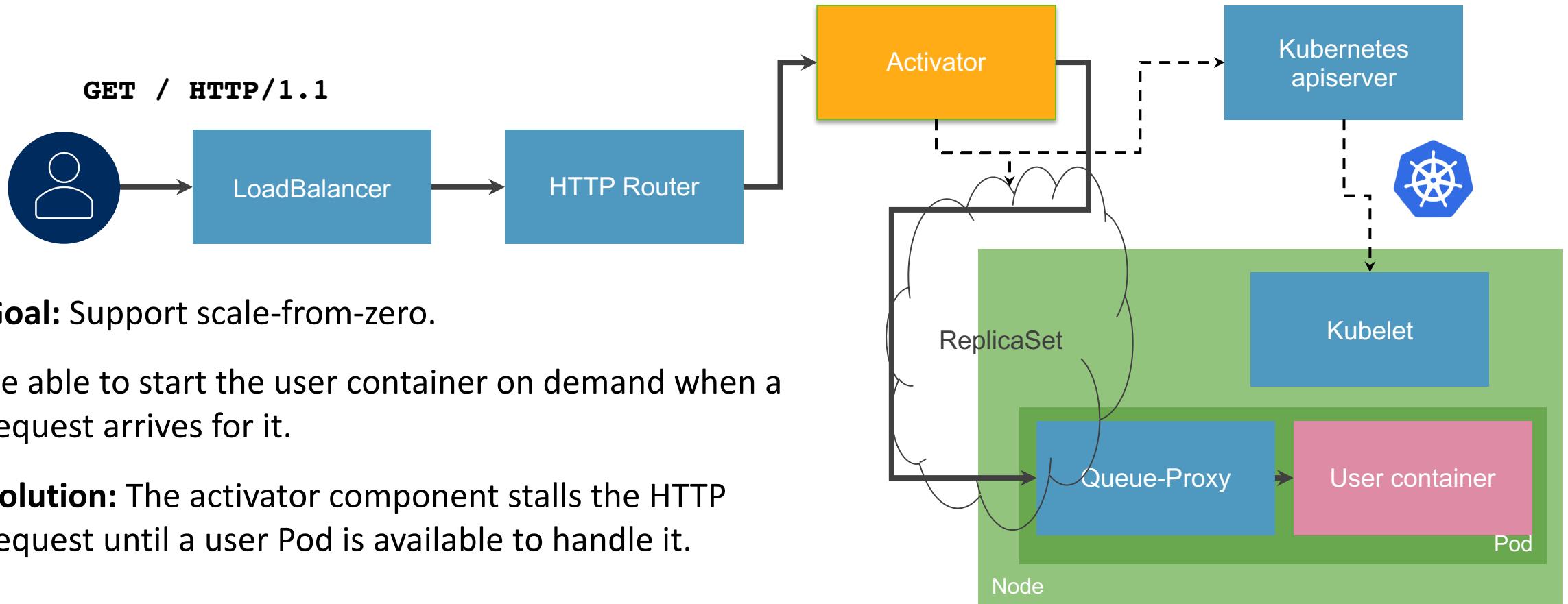
Goal: Support bounded concurrency.

AWS Lambda runs a max of one request per process. This means many FaaS functions aren't written to be threadsafe!

Solution: Add queue-proxy container to limit concurrency of requests to the user container (optional; high concurrency is also supported).



Life of a query

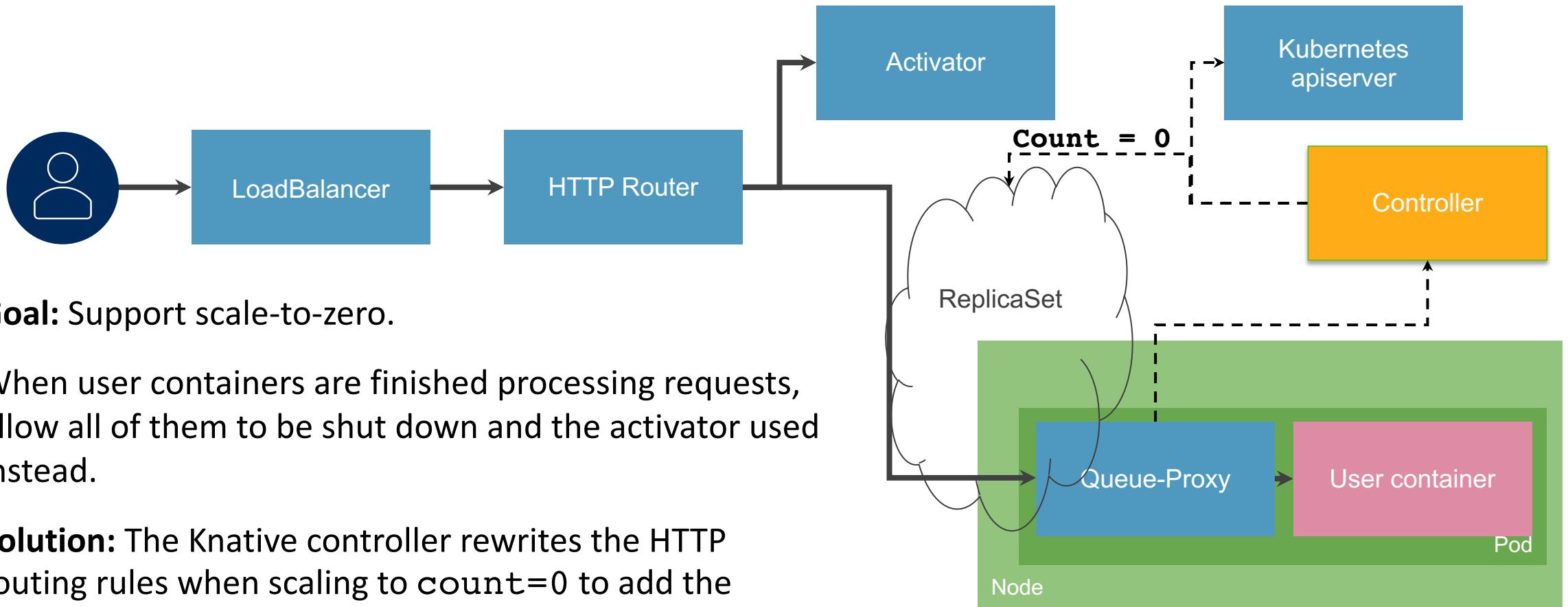


Goal: Support scale-from-zero.

Be able to start the user container on demand when a request arrives for it.

Solution: The activator component stalls the HTTP request until a user Pod is available to handle it.

Life of a query

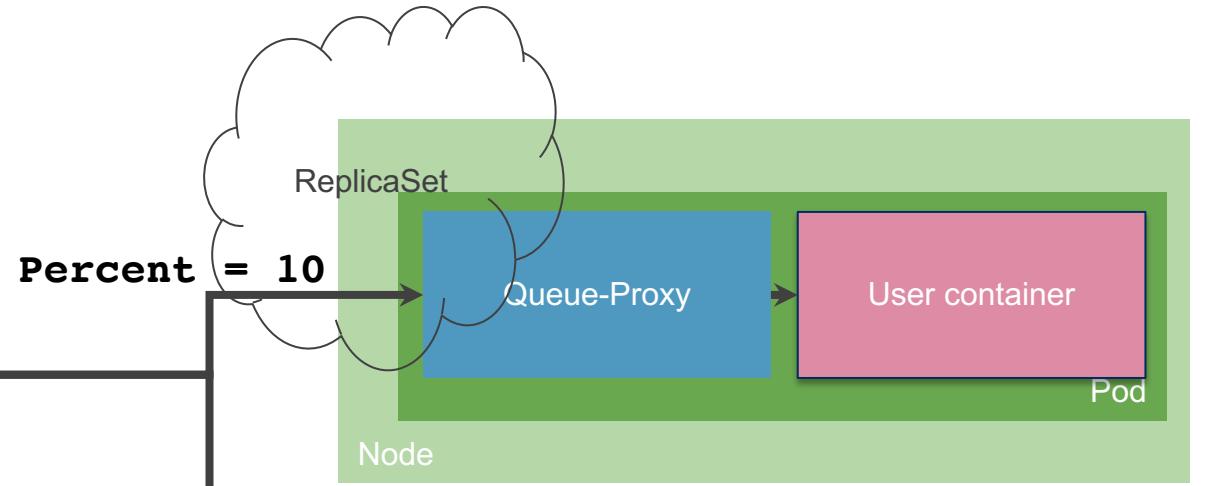


Goal: Support scale-to-zero.

When user containers are finished processing requests, allow all of them to be shut down and the activator used instead.

Solution: The Knative controller rewrites the HTTP routing rules when scaling to count=0 to add the Activator to the Kubernetes service.

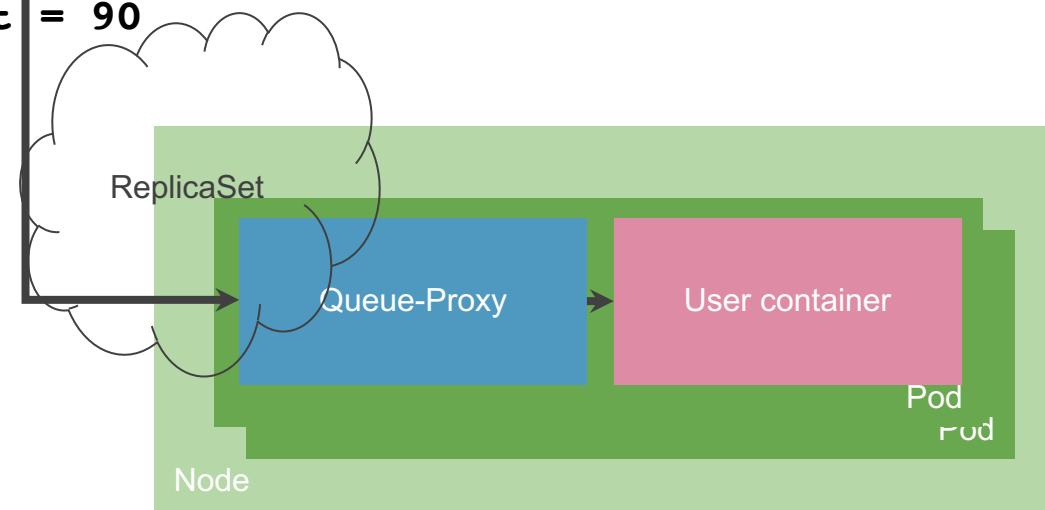
Life of a query



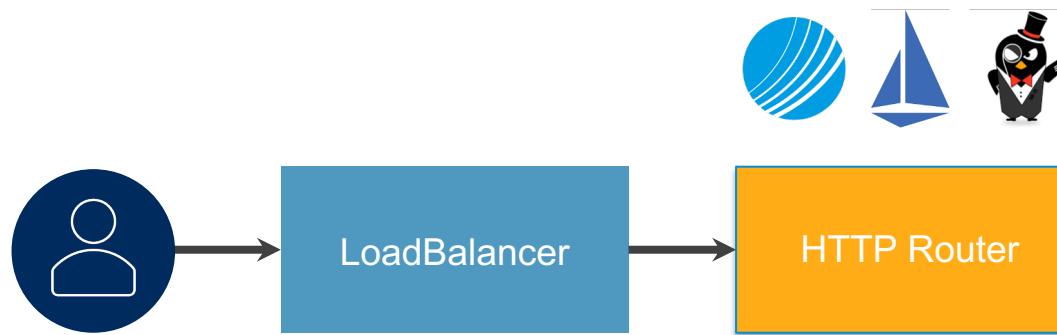
Goal: Support percentage rollout of new features.

Small services may only receive enough traffic to use one or two Pods. This makes it difficult to do a good progressive rollout.

Solution: Knative supports percentage-based traffic assignment to Revisions of a Service.



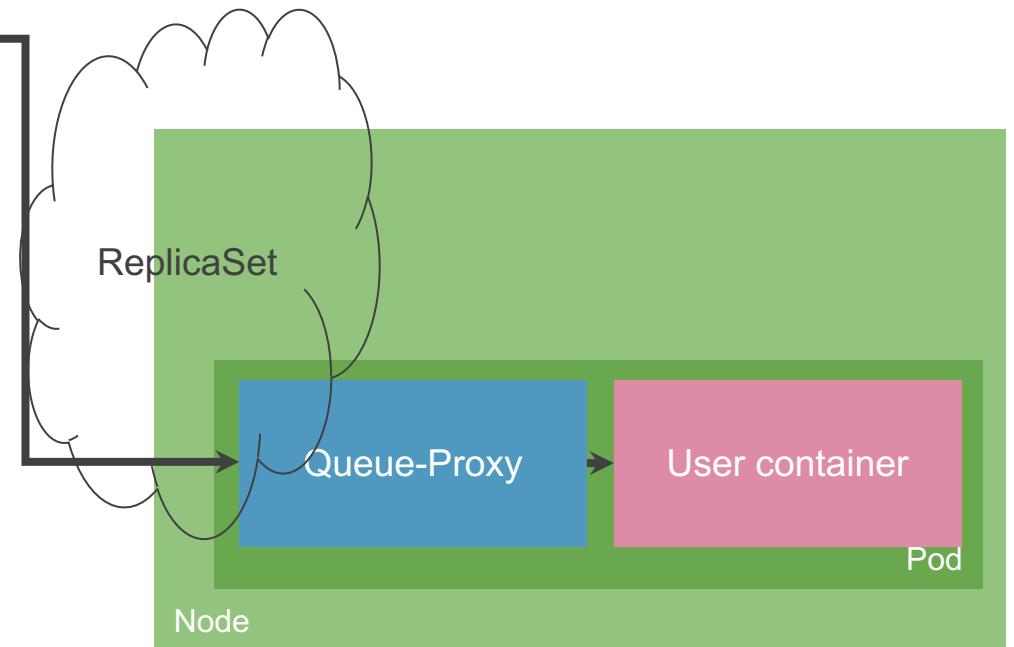
Life of a query



Goal: Support multiple HTTP routers.

Initial release supported only Istio, but community feedback indicated demand for other HTTP routers.

Solution: Implemented a custom interface for HTTP routers that meet minimum requirements (percent splits, header rewrite).



Life of a Service

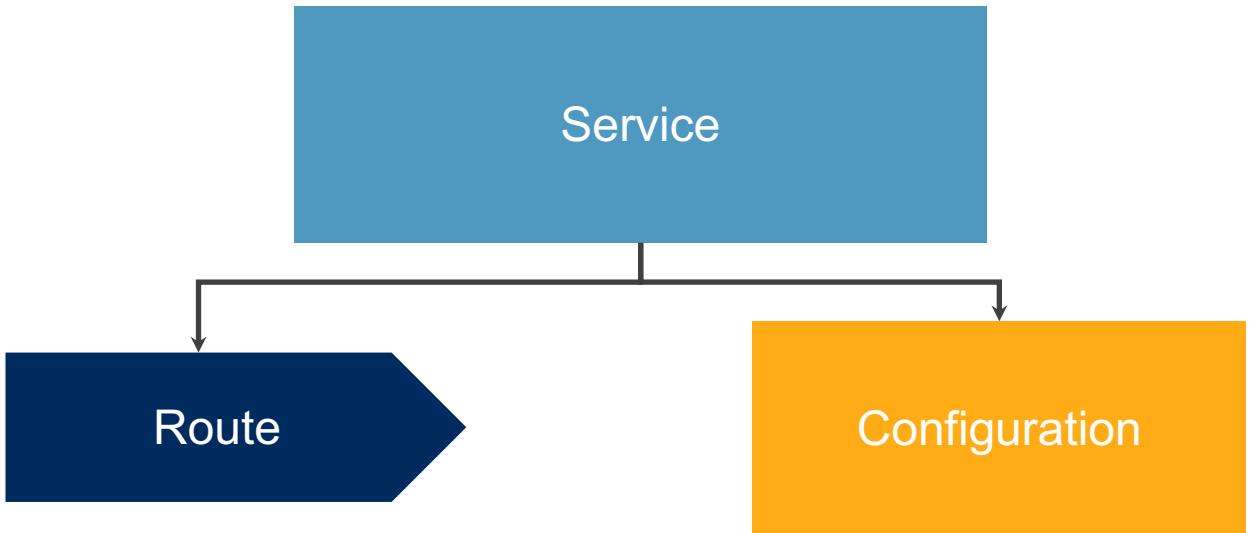
Knative Serving, part 2

Control Plane



Life of a Service

- **Service** is the top-level object that most developers will interact with.
- It controls Route and Configuration objects in a 1:1 relationship.

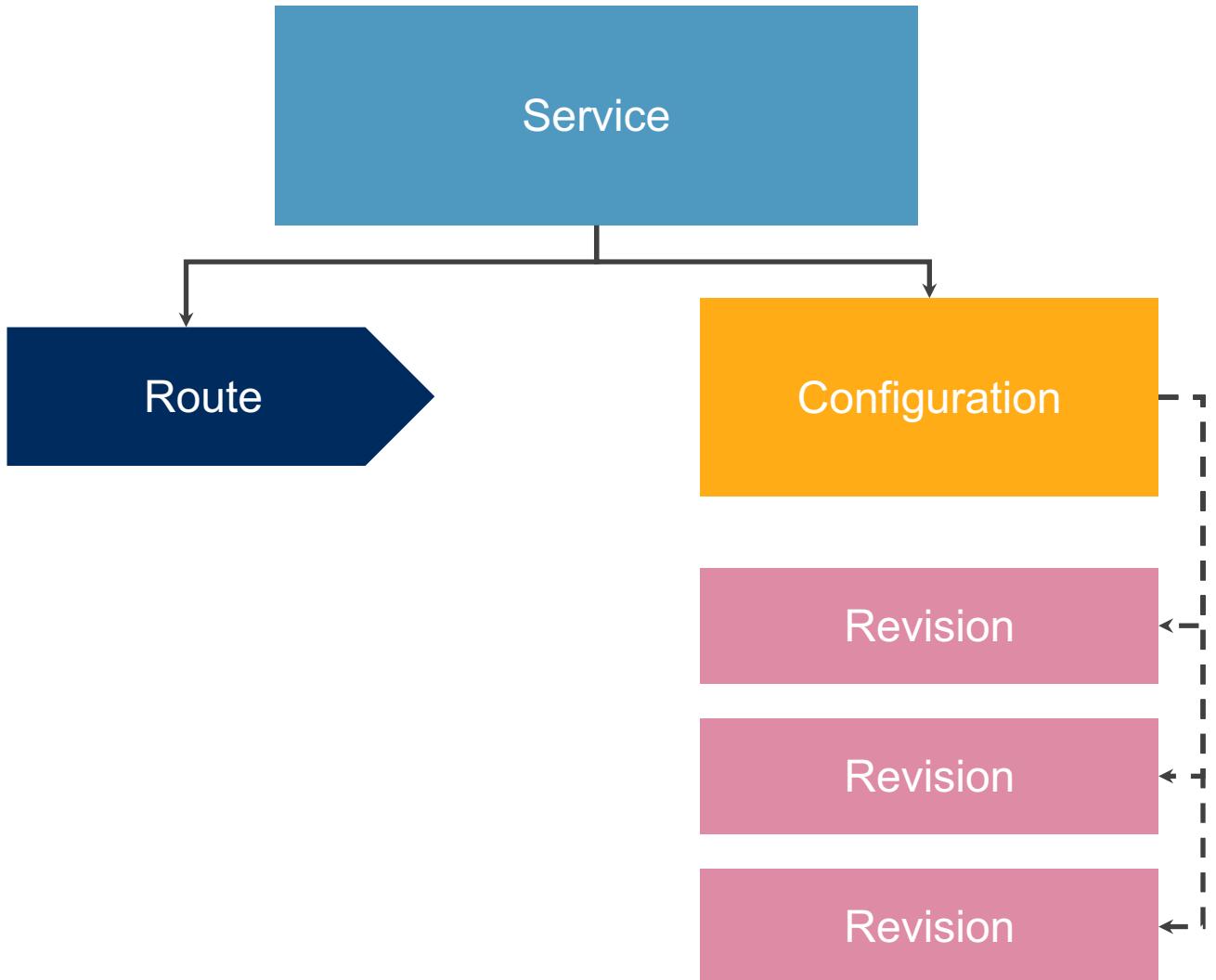


```
apiVersion: serving.knative.dev/v1
kind: Configuration
metadata:
  name: cat-pictures
  namespace: meow
spec:
  template:
    spec:
      containers:
        - image: meow/mix:v3
```

Life of a Service

- **Configuration** manages the desired state of running containers.
- It works like a “HEAD” of version control, and creates Revisions for each change to the Configuration.

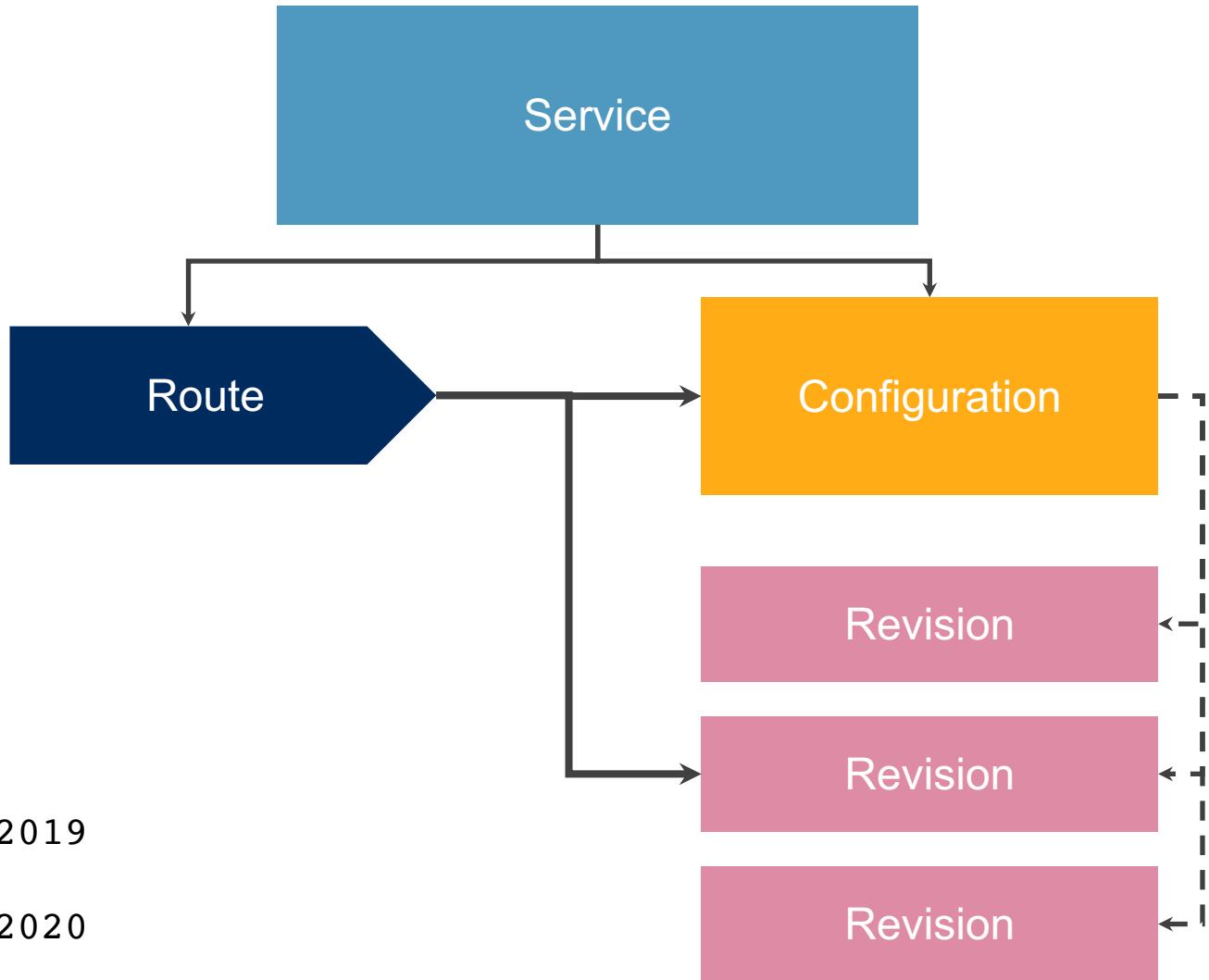
```
apiVersion: serving.knative.dev/v1
kind: Configuration
metadata:
  name: cat-pictures
  namespace: meow
spec:
  template:
    spec:
      containers:
        - image: meow/mix:v3
```



Life of a Service

- **Route** controls the distribution of traffic to Revisions.
- Route supports percentage traffic distribution to named revisions or to the latest Revision.

```
apiVersion: serving.knative.dev/v1
kind: Route
metadata:
  name: cat-pictures
  namespace: meow
spec:
  traffic:
    - revisionName: cat-pictures-12-14-2019
      percent: 90
    - revisionName: cat-pictures-01-09-2020
      percent: 10
```



Life of a Service

- **Revision** represents a snapshot of the state of a Configuration.
- Revision exists to enable rollback and tracking changes over time.

```
apiVersion: serving.knative.dev/v1
kind: Revision
metadata:
  name: cat-pictures-b7g8d
  namespace: meow
spec:
  template:
    spec:
      containers:
        - image: meow/mix:v3
```

