Serverless Operations

Soam Vasani

fission.io

fission

Fission: Serverless Functions

- Open source Kubernetes-native FaaS framework
- Lambda-like service both on-premise and in the cloud
- Designed to be easy to use, productive and fast
- Tunable cost/performance tradeoffs

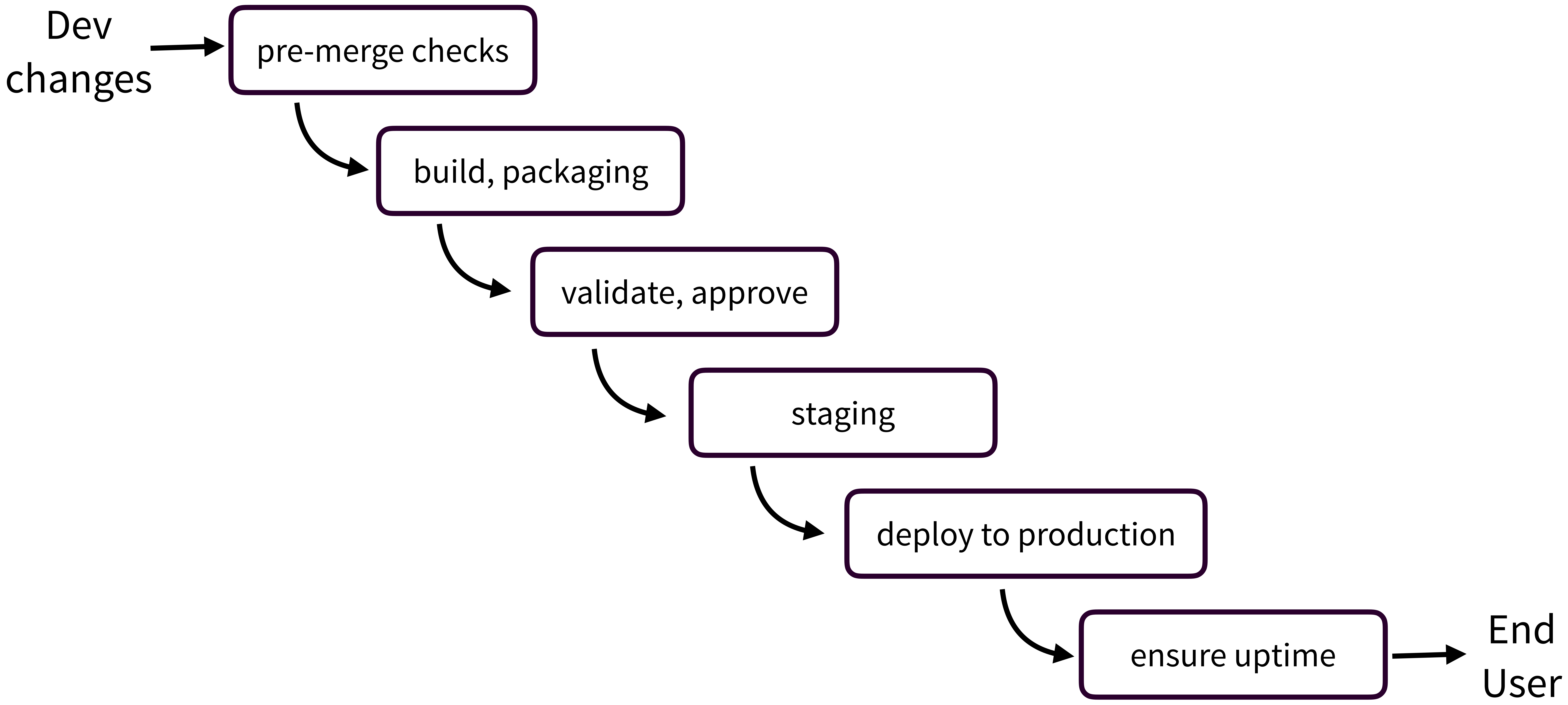
Why Serverless

- Developer productivity: focus on application code
- Pay for what you use, idle = free
- Will occupy an important part of the software stack in the future
- On-premise benefits!

Production-ready Serverless Apps

- Serverless will exist in various forms in modern infrastructure
  - FaaS in the cloud and on-premise
  - As cloud services (Lambda etc) and on Kubernetes
- We want the productivity advantages — but we want to go **faster, safely** and at **scale**

The DevOps Pipeline



Serverless Operations from Dev to Production

Some best practices and patterns:

1. Declarative configuration
2. Live-reload for fast feedback
3. Record-replay for testing and debugging
4. Canary Deployments
5. Monitoring with metrics and tracing
6. Cost optimization

1

Specifying Applications

Spoiler Alert: Use declarative configuration!

Specifying Apps: Declarative Config

- Specify app source, packaging, and configuration as a series of configuration files, rather than imperative scripts
- **Imperative**: “Copy this file there and run it”
- **Declarative**: “Ensure this file exists and that it’s running”

Benefits of Declarative Config

- Now that we’ve specified our app declaratively, we can:
  - Do better validation before deploys
  - Do one-click deploys
  - Deploy without worrying about current state of the cluster: the system will find differences and reconcile them. Great for upgrades!
  - Version everything in Git: Collaborate, auto-deploy, rollback. “Gitops”
  - Watch files and “live-test” your code

Declarative Config in Fission

- Fission resources (Functions, Environments, Triggers) are Kubernetes ***Custom Resources*** *(*CRDs*)*, so they can be stored as YAML/JSON files
- Fission automatically generates initial config: **Never write YAML from scratch**
  - `` `fission function create --spec …` ``
- Also specify **packaging**: how local files get packaged and uploaded

Deploying with Declarative Config

- **`fission spec validate`**
  - `Checks for consistency and common errors`
- **`fission spec apply`**
  1. **Packages** source code
  2. **Uploads** to cluster
  3. **Builds**, gathers dependencies (if necessary)
  4. **Creates/Upgrades/Deletes** Fission Kubernetes resources

2

Live-Reload

Fast feedback means fewer bugs

Live-reload: Test as You Type

- The sooner you find the problem, cheaper it is to fix
- Accelerating feedback loops improves quality
- “Live-reload” means code is instantly deployed into a test system as soon the developer is saving their files
- Instant feedback on whether the change is correct

Live-reload in Fission

- `` `fission spec apply `` **`--watch`** `` ` ``
- Save your file, fission deploys it to a test cluster automatically within 1-5 seconds
- Because you’re testing on a real cluster, you can mimic your real deployment more closely
- This gives you very quick feedback on whether your changes are correct

3

Record-Replay

Reproducing bugs is the easiest way to get them fixed

Record-Replay

- Record-replay is a technique for saving the events that invoked a function and simulating these events at a later point for testing or debugging
- **Testing**: Replay a request to test if a new version of a function behaves like the old one: regression testing
- **Debugging**: Inspect execution of a function on a past input

Record-Replay Use Cases

- **Dev** can use Recording during testing to make sure we can reproduce a failure
- **Ops** can enable recording on a subset of production traffic, to enable devs to reproduce problems, debug them, and verify updated versions

Record-Replay in Fission

- Fission has built-in record-replay, which can store HTTP requests and responses, and replay on demand
- Fission lets you create “recorder” resources for functions, which configure what is recorded and how long it’s retained
- Replay requests on demand, either on a new version or with a debugger on the old version

4

Canary Deployments

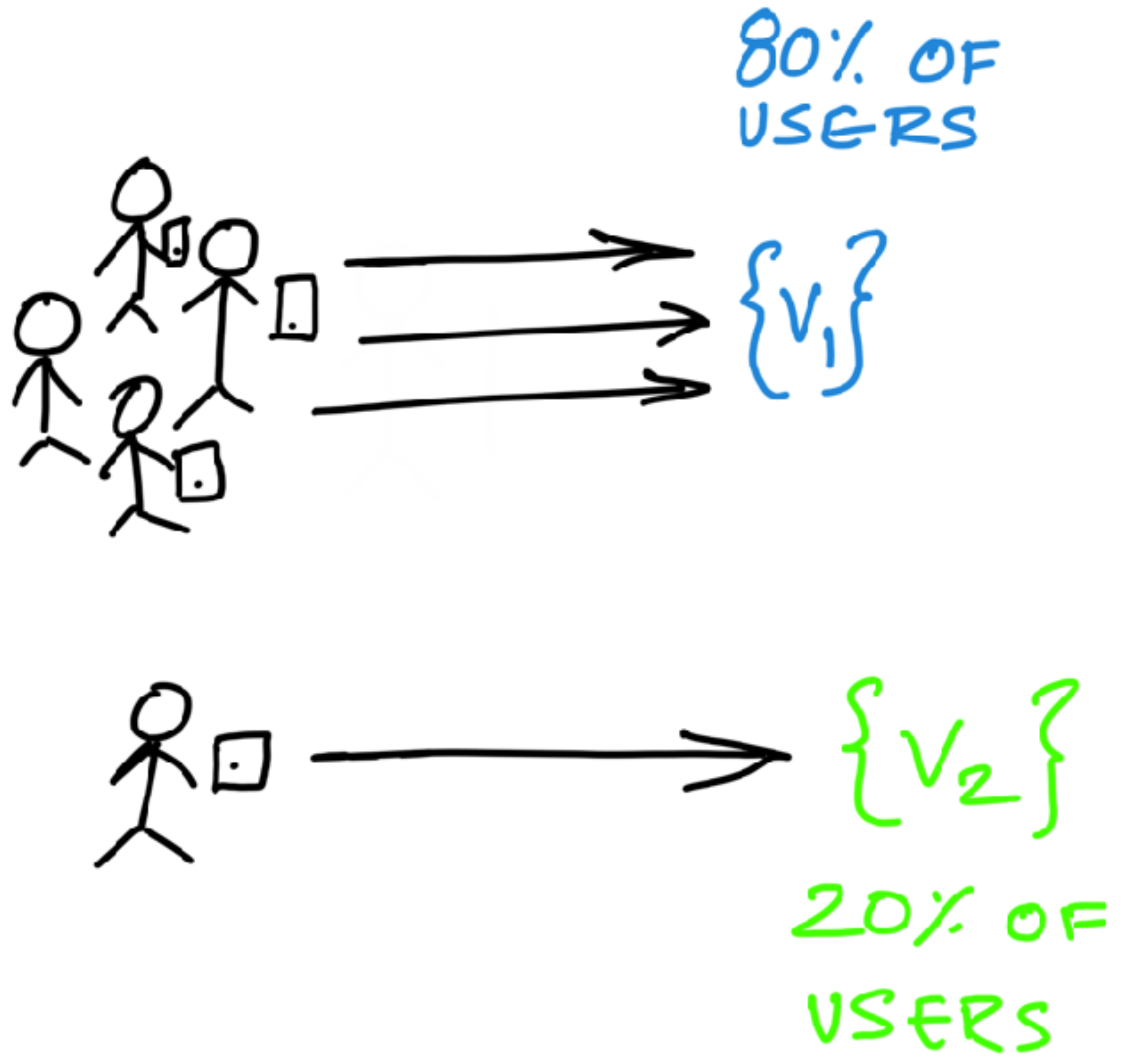
Reduce risk by slow, careful roll out of new versions

Reducing the Risk of Failed Deployments

- After all testing is done, deployment to production is still risky
- Test and Staging environments are never quite the same as production
- After a version is qualified in testing, a good strategy is to deploy *incrementally*
- For example, 10% of your users get the new version, and if all goes well you gradually increase that percentage.

Canary Deployments

- Let's say we have version **V1** deployed
- We've tested version **V2** and are ok with it in testing
- Now we deploy version **V2** but only send 20% of users to it
- This is a *canary deployment* — we proceed with the rollout only if the new version works well on the 20%



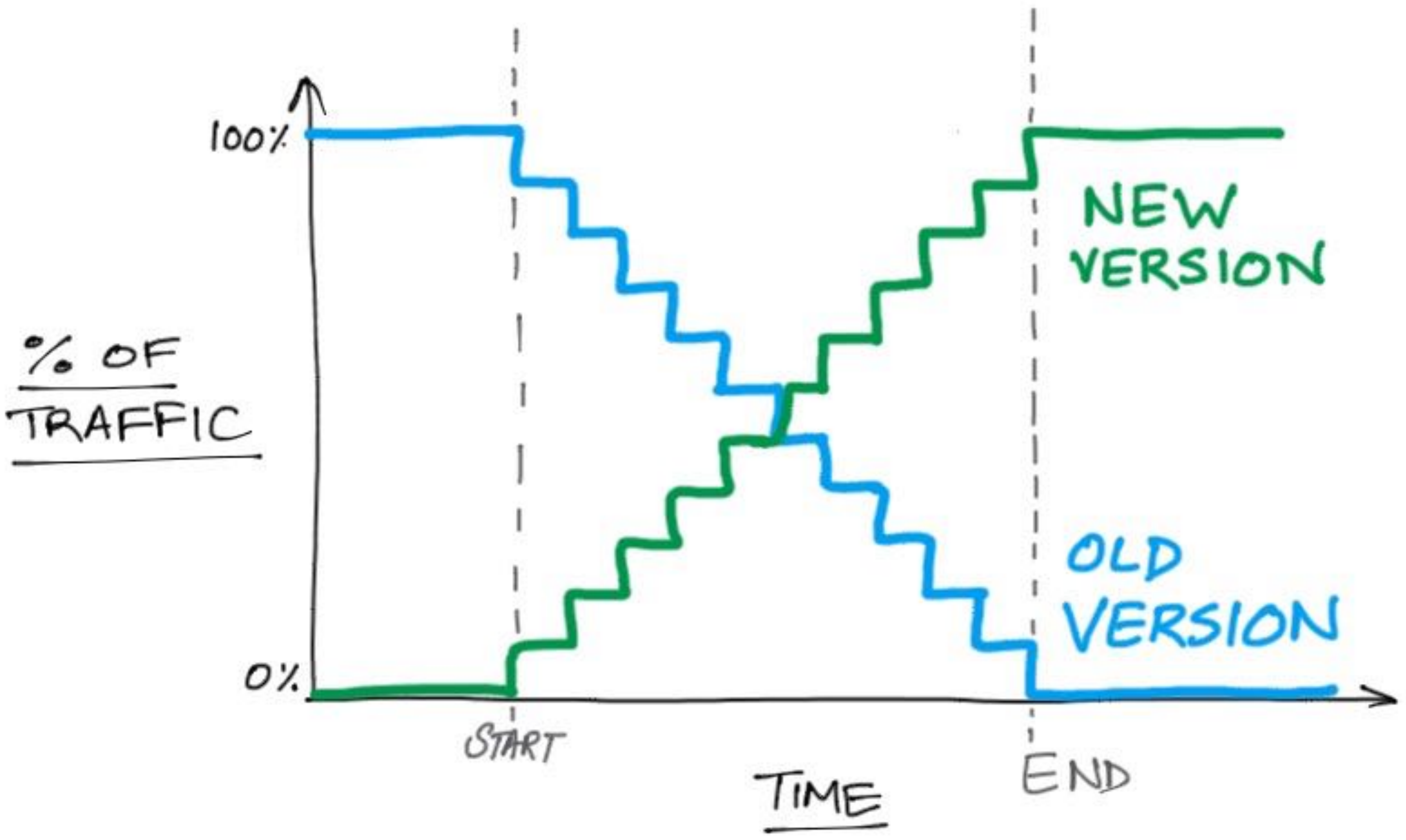
Automating Canary Deployments

- With Canary Deployments you have to monitor for success of the canary, and decide whether to go ahead with the deployment
- In a FaaS system, we know whether a function succeeded or failed
- We can automate the process of rolling forward or rolling back

Automated Canaries in Fission

- Fission has built-in automated canary deployments. They can be configured with:
  - The fraction of traffic for the new version
  - The error rate that we call a failure
  - The rate at which to “grow” the new version as long as it’s succeeding
  - The function is rolled back at any point if it does not succeed

Traffic Graph in Canary Deployments



5

Metrics, Tracing, Logs, Alerts

Understand your systems performance

Monitoring Serverless Systems

- Many aspects: logs, metrics, alerts, tracing
- **Log Aggregation** using fluentd — save them somewhere searchable (e.g. Elastic stack)
- **Metrics**: Use Prometheus
  - Prometheus has Alertmanager which can be used for **alerts** based on metrics
- **Tracing**: Use Jaeger or other OpenTracing implementations

Fission Metrics

- Fission automatically tracks timing and success rate metrics for all functions
  - Function run time, fission overheads, error codes
- Fission has Prometheus integration for metrics collection
- You can build dashboards with Grafana, and alerts with Prometheus Alertmanager

6

Cost Optimization

Balance performance and cost in the cloud and the datacenter

Cost Optimization

- Most systems have cost/performance tradeoffs
- Public cloud serverless lets you pay for what you use, though the tradeoffs get worse as usage gets higher
- In the on-premise you still care about utilization — resources used should be proportional to actual demand, so they are available for other services that may need them

Cost Optimization

- Big topic!
- On public cloud, clever use of **Reserved Instances**, cheaper Spot/Pre-emptible Instances can yield significant savings
- Careful configuration of **resource limits** for applications in a cluster
- On all infrastructures, **autoscaling** can make clusters more efficient — growing resource utilization only when there is demand and shrinking it otherwise

The Cold-Start Problem

- Ideally, services with **zero usage should be free**
- But services should be able to **start quickly** when there is demand for them
- This is the ***cold-start* problem**: how do we ensure low idle costs while simultaneously providing low latency?

Cold Starts in Fission

- Built-in **cold-start optimization**: use a *pool* of pre-warmed containers
- Pool size can be configured; the cost of the pool is amortized over all functions in the cluster
- When they are invoked, functions are loaded into a container from the pool
- Functions can also be configured not to use a pool at all, slowing them down but further reducing cost

Cost Optimizations in Fission

- Function execution is **tunable**: choose a point on the cost/performance tradeoffs
- Not subject to lambda pricing model — can be as cheap as the cheapest VM instance (RI, spot etc.)
- On-premise usage can be a cost savings, especially if you have existing infra

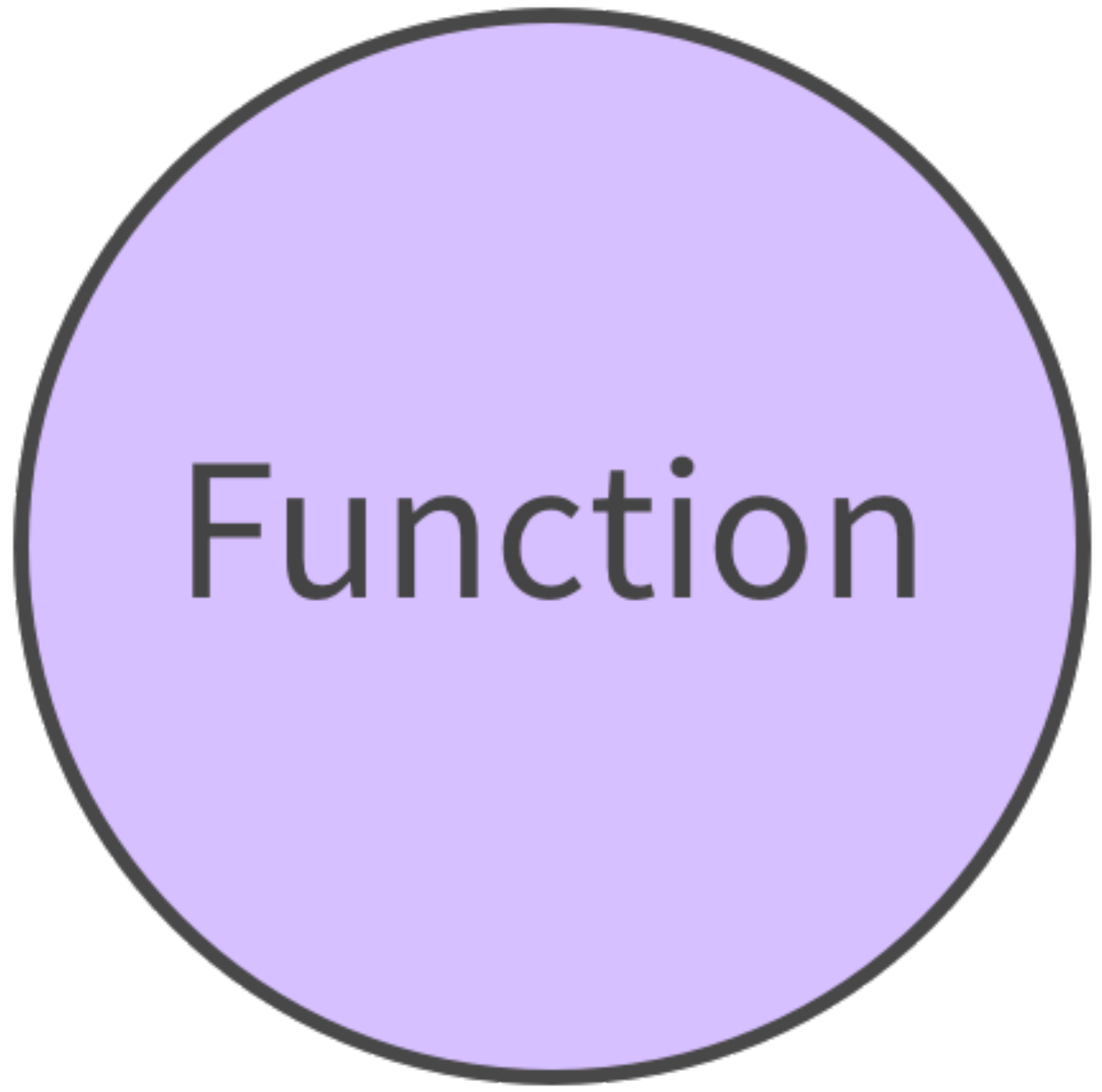
Cost Optimizations in Fission

- Configure CPU and memory resource usage limits for functions
- Configure autoscaling parameters: min and max scale, target CPU utilization

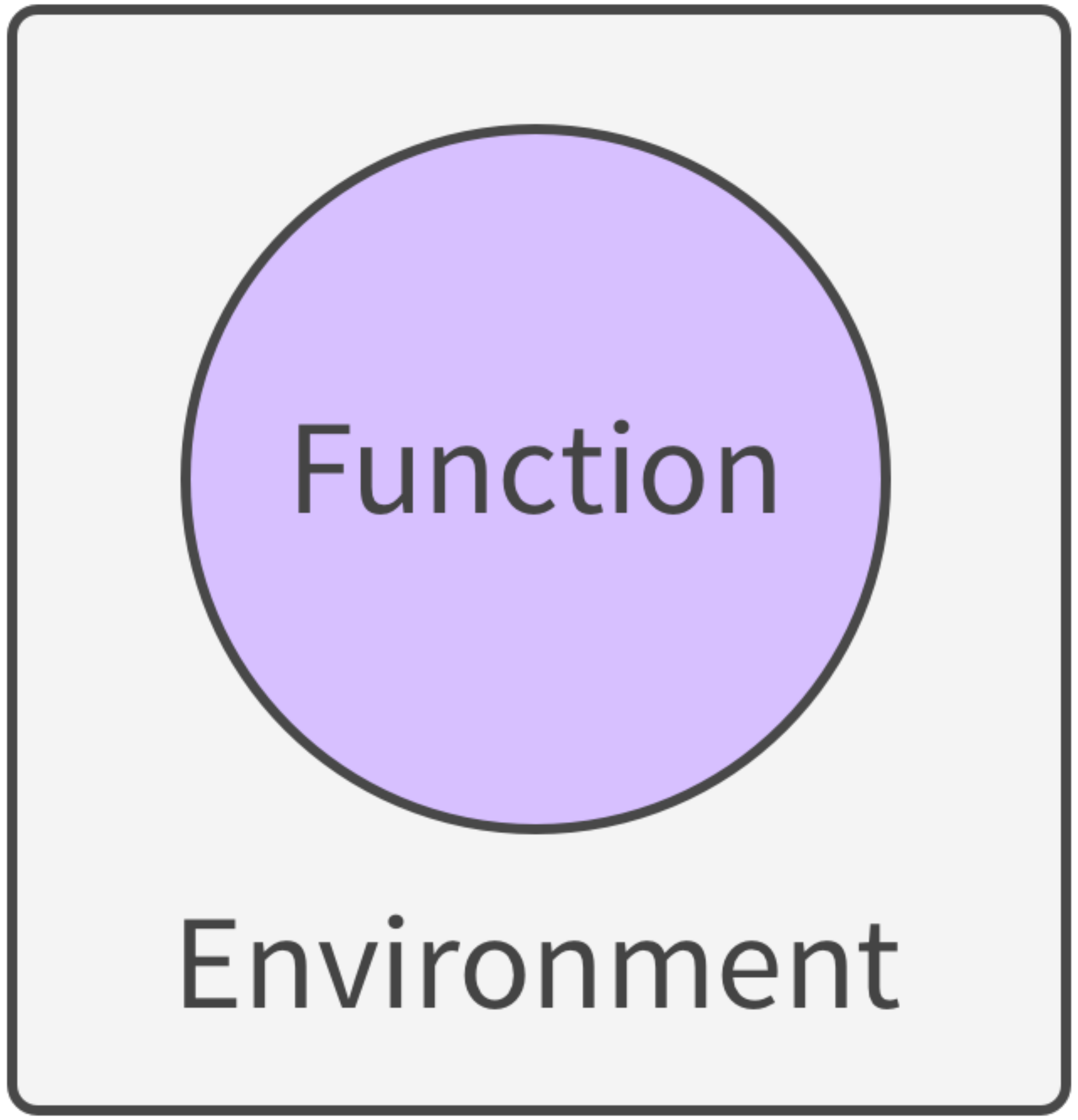
Demo!

- Hello, world
- Declarative config
- Live reloads
- Record-replay
- Canary deployments — we’ll also metrics in prometheus

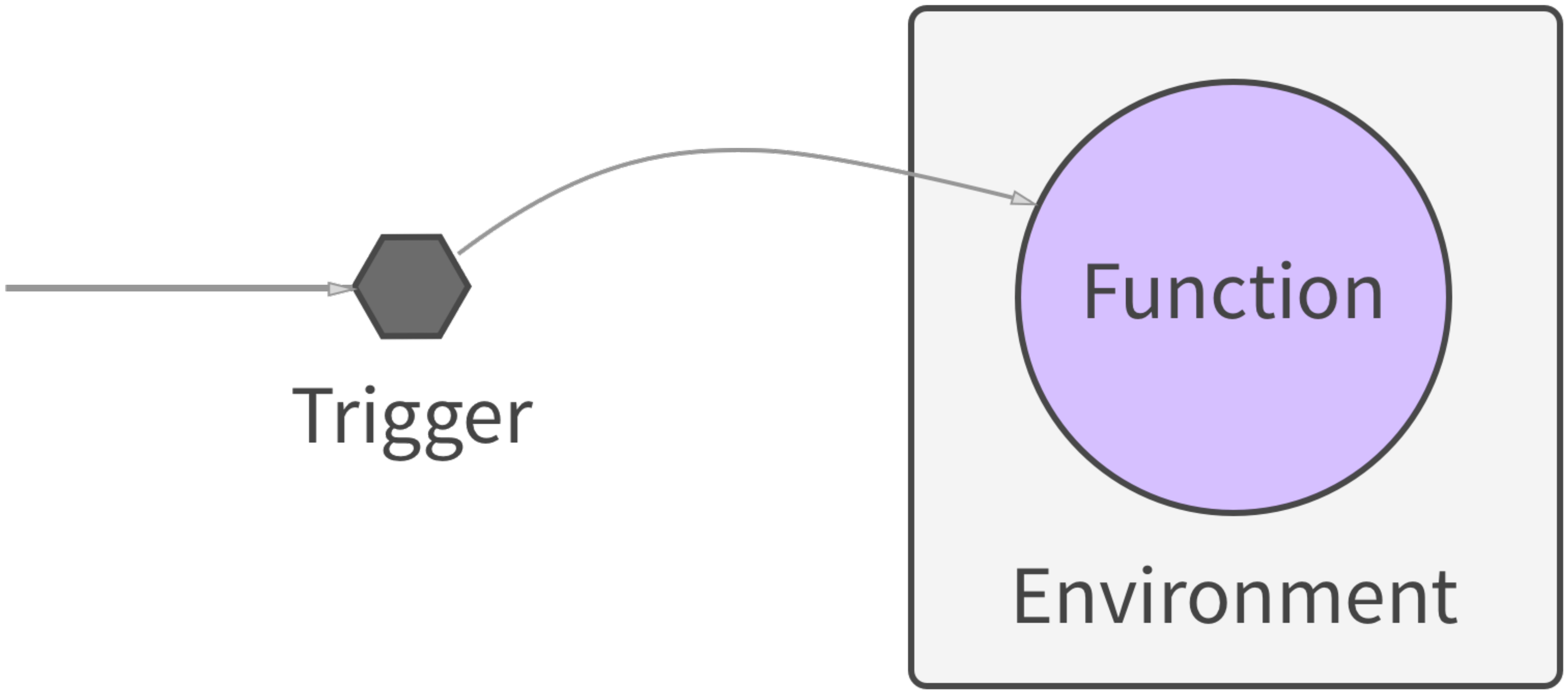
Get Started with Fission



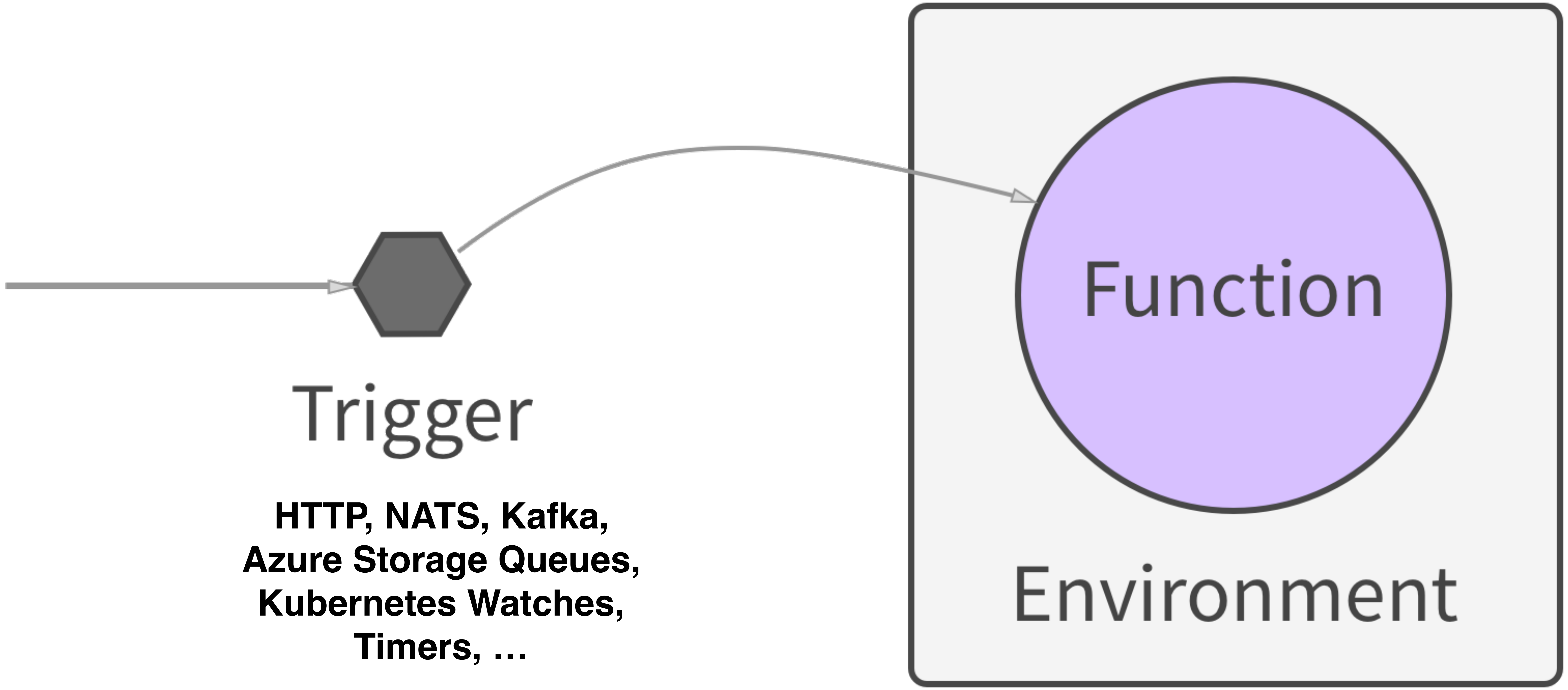
Get Started with Fission



Get Started with Fission



Get Started with Fission



HTTP, NATS, Kafka, Azure Storage Queues, Kubernetes Watches, Timers, …

NodeJS, Python, Go, Ruby, C#, PHP, Bash, Perl, Java

Get Started with Fission

- Visit: **fission.io**
- github.com/fission/fission — see milestones for upcoming features
- **Install** latest release 0.10: docs.fission.io/latest/installation/
- Canaries coming in Fission 0.11
- **Slack**: slack.fission.io — Ask us anything!
- **Twitter**: @fissionio