Orchestrating Containers with Consul and Terraform

Mitchell Hashimoto

@mitchellh

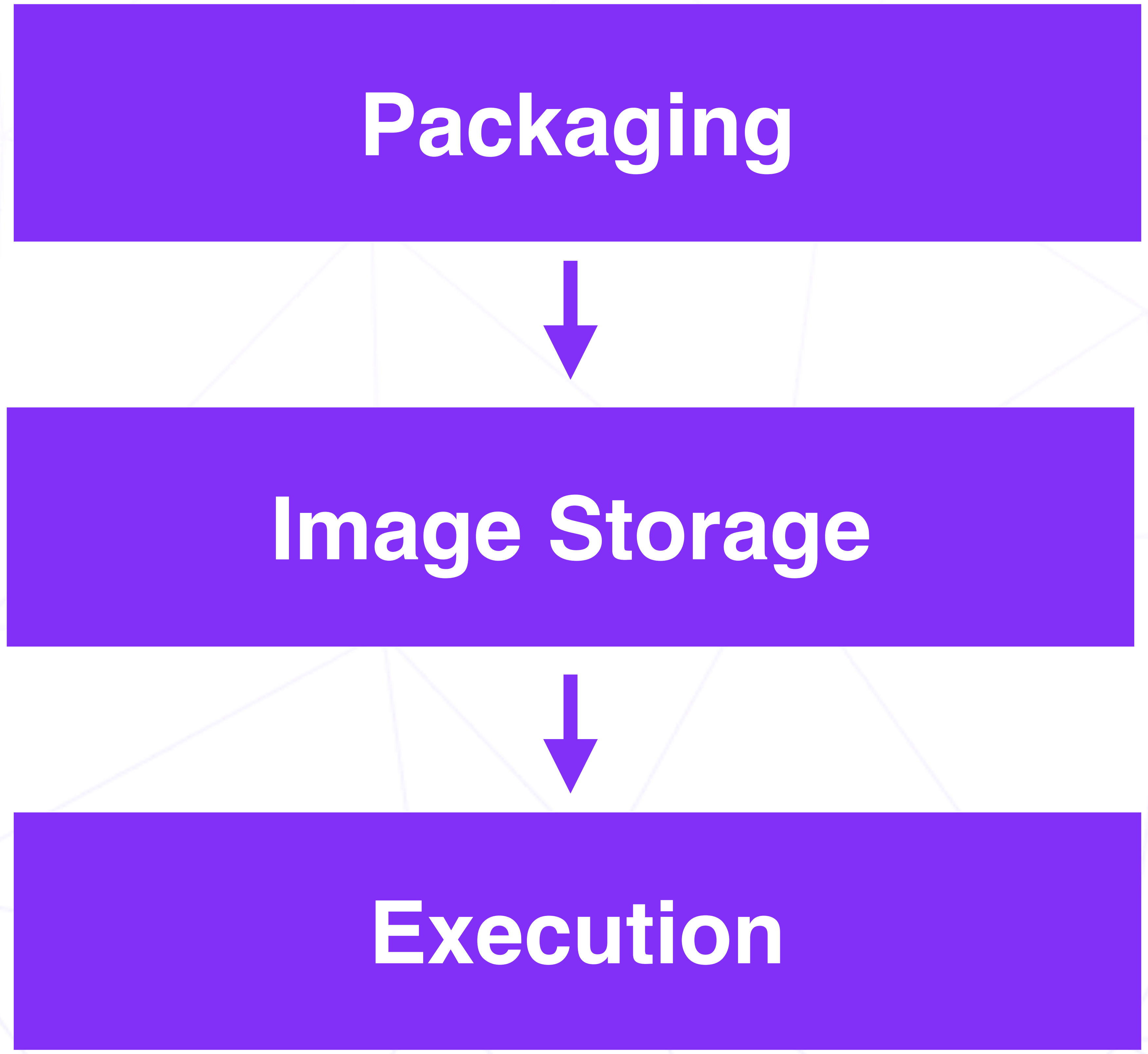
ORCHESTRATION?

Why is it needed? What is it?

ORCHESTRATION

- Do some set of actions, to a set of things, in a set order.
- **Ultimate goal:** safely deliver applications at scale

PROBLEMS CONTAINERS SOLVE

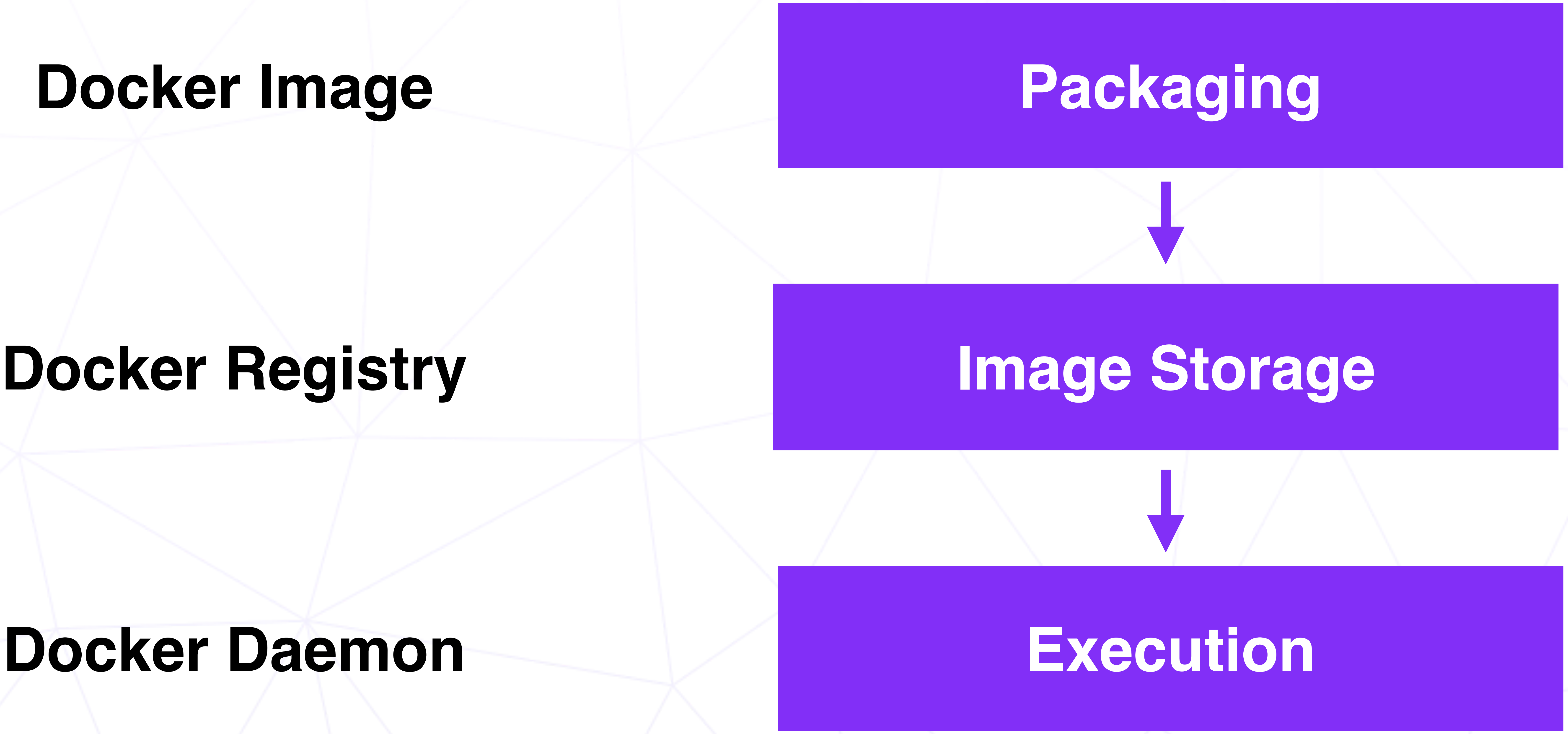


Packaging

Image Storage

Execution

PROBLEMS CONTAINERS SOLVE



Docker Image → Packaging

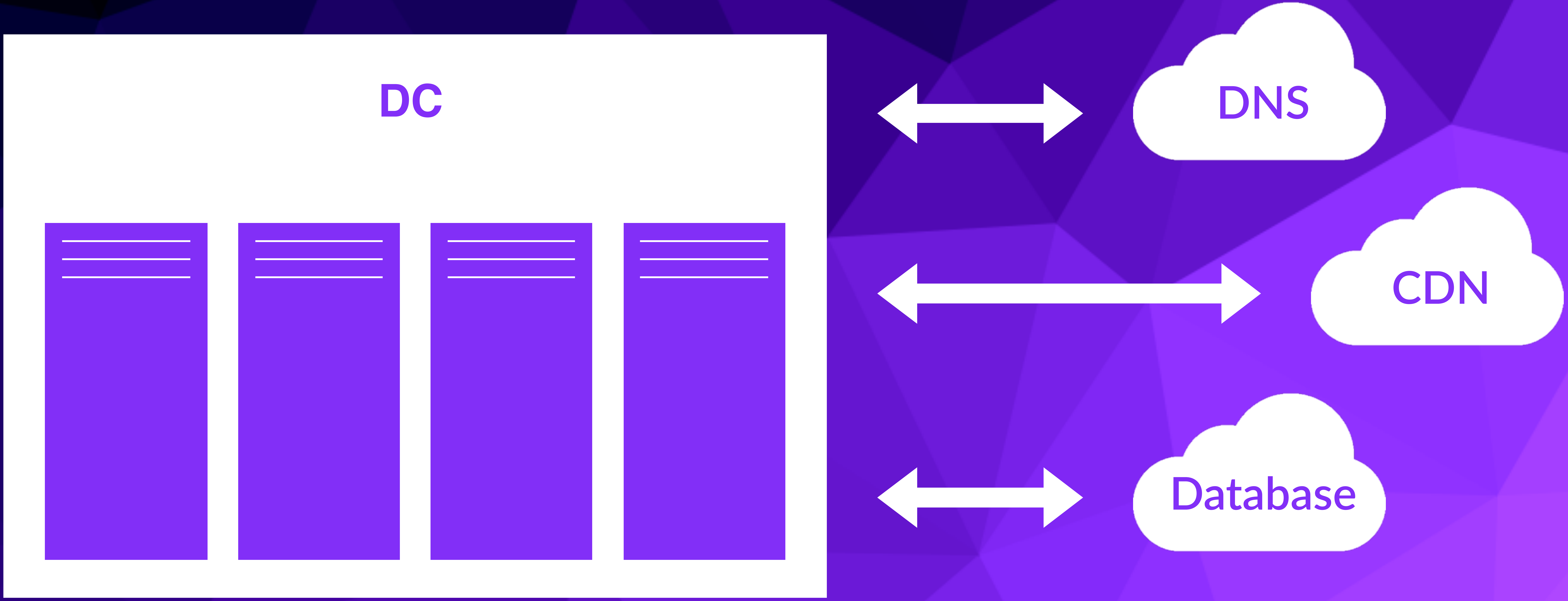
Docker Registry → Image Storage

Docker Daemon → Execution

A LOT OF OTHER PIECES

- Infrastructure lifecycle and provisioning
- Monitoring
- Service discovery
- Service configuration
- Security/Identity
- Deployment and application lifecycle

INFRASTRUCTURE



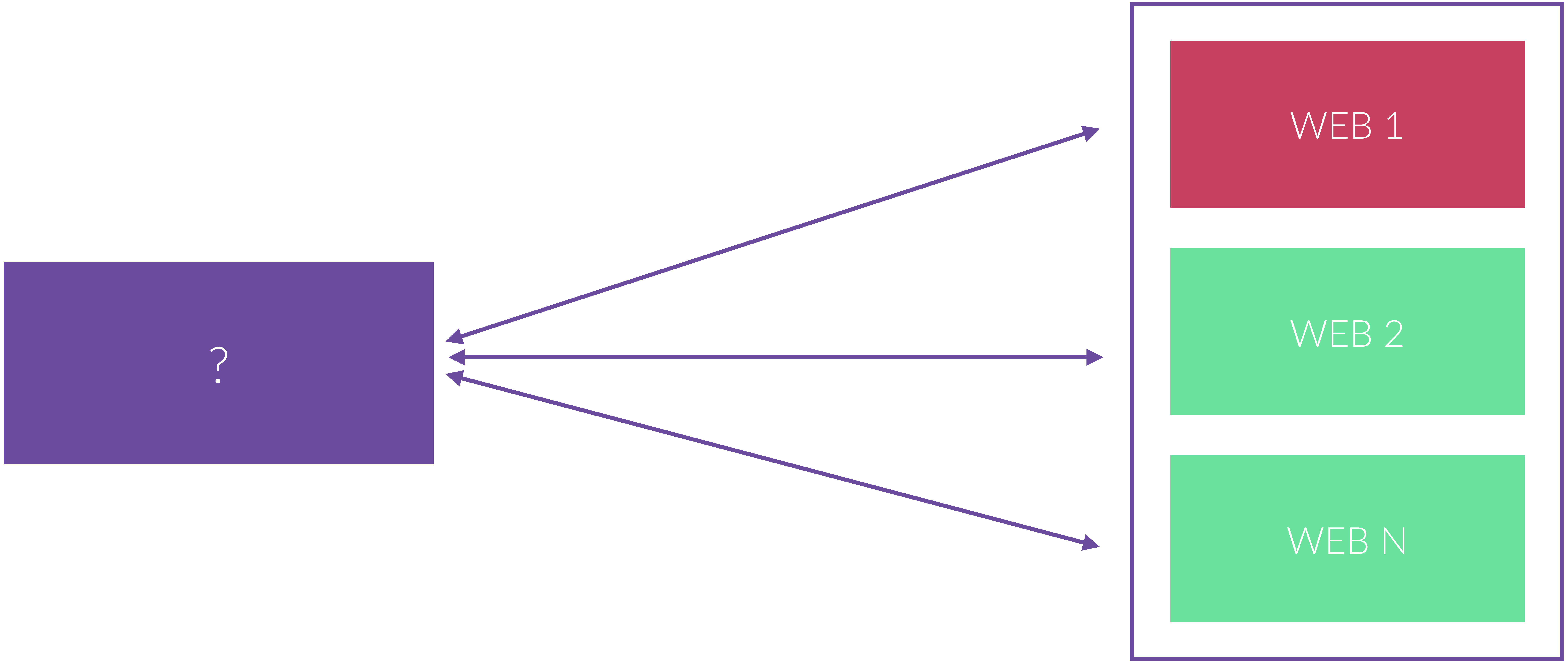
INFRASTRUCTURE

- Container hosts
- Storage
- Network
- External services

INFRASTRUCTURE

- Creation, update, destroy
- Creation is easy
- Update is hard
- Update with minimal downtime is hardest
- Has its own lifecycle events: canary infrastructure changes, rolling, etc.

MONITORING



MONITORING

- Level of monitoring: node, container, service
- Propagation of information
- Utility of the information in other orchestration actions

SERVICE DISCOVERY AND CONFIG

- Where is service *foo?*
- Runtime configuration of a service (especially in an immutable world)
- All of the above at the speed of containers

SECURITY

- Identity for service to service communication
- Storage and retrieval of secrets

APPLICATION LIFECYCLE

- Canary, Rolling, Blue/Green
- Create before destroy
- Triggering a deploy (communication)
- Monitoring a deploy

LIVING WITH LEGACY

- Non-container to container isn’t atomic
- Orchestration needs to include non-containerized systems
- Time period for this is probably years
- What about a post-container world?

AN OLD PROBLEM

It all should sound familiar

AN OLD PROBLEM

- “Orchestration problems” not caused by containers
- Higher density/speed reveals and exacerbates problems
- New aspects: public cloud, growing external service footprint
- These orchestration problems existed yesterday, exist today, and will exist tomorrow, in slightly different forms

SOLUTIONS TO LAST

Infrastructure lifecycle, service discovery, monitoring, and orchestration at scale for all infrastructures.

terraform.io

Build, combine, and launch infrastructure safely and efficiently.

terraform.io

What If I asked you to…

- create a completely isolated second environment to run an application (staging, QA, dev, etc.)?

- deploy or update a complex application?

- document how our infrastructure is architected?

- delegate some ops to smaller teams? (Core IT vs. App IT)

What If I asked you to…

- create a completely isolated second environment to run an application (staging, QA, dev, etc.)? **One command.**
- deploy a complex new application? **Code it, diff it, pull request.**
- update an existing complex application? **Code it, diff it, pull request.**
- document how our infrastructure is architected? **Read the code.**
- delegate some ops to smaller teams? (Core IT vs. App IT) **Modules, code reviews.**

Terraform

- Create infrastructure with code: servers, load balancers, databases, email providers, etc.

- One command to create, update infrastructure.

- Preview changes to infrastructure, save diffs.

- Use code + diffs to treat infrastructure change just like code change: make a pull request, show the differences, review it, and accept.

- Break infrastructure into *modules* to encourage/allow teamwork without risking stability.

Infrastructure as Code

DigitalOcean Droplet with DNS in DNSimple

```
resource "digitalocean_droplet" "web" {
    name = "tf-web"
    size = "512mb"
    image = "centos-5-8-x32"
    region = "sfo1"
}

resource "dnsimple_record" "hello" {
    domain = "example.com"
    name = "test"
    value = "${digitalocean_droplet.web.ipv4_address}"
    type = "A"
}
```

Infrastructure as Code

DigitalOcean Droplet with DNS in DNSimple

```
resource "digitalocean_droplet" "web" {
    name = "tf-web"
    size = "512mb"
    image = "centos-5-8-x32"
    region = "sfo1"
}

resource "dnsimple_record" "hello" {
    domain = "example.com"
    name = "test"
    value = "${digitalocean_droplet.web.ipv4_address}"
    type = "A"
}
```

Infrastructure as Code

DigitalOcean Droplet with DNS in DNSimple

```
resource "digitalocean_droplet" "web" {
    name = "tf-web"
    size = "512mb"
    image = "centos-5-8-x32"
    region = "sfo1"
}

resource "dnsimple_record" "hello" {
    domain = "example.com"
    name = "test"
    value = "${digitalocean_droplet.web.ipv4_address}"
    type = "A"
}
```

Infrastructure as Code

DigitalOcean Droplet with DNS in DNSimple

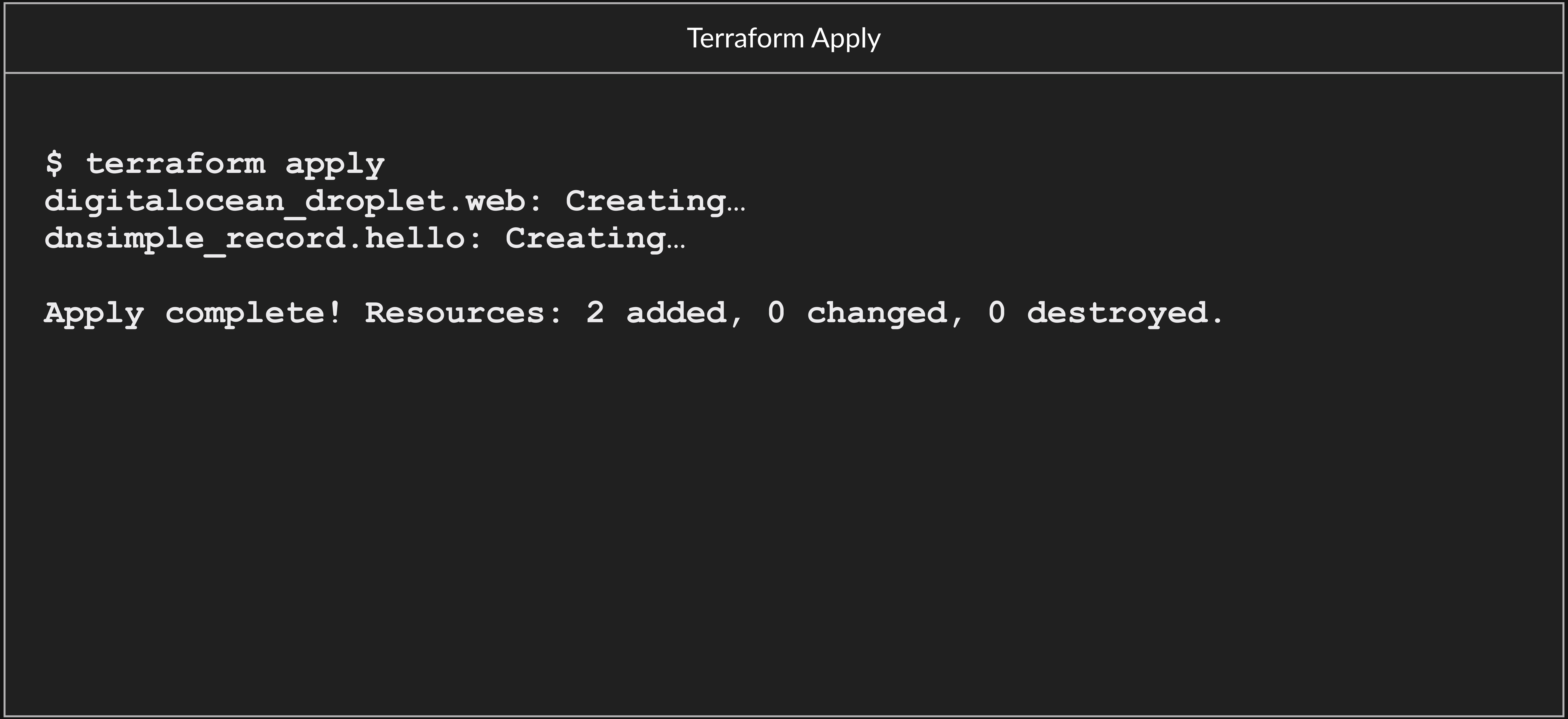
```
resource "digitalocean_droplet" "web" {
    name = "tf-web"
    size = "512mb"
    image = "centos-5-8-x32"
    region = "sfo1"
}

resource "dnsimple_record" "hello" {
    domain = "example.com"
    name = "test"
    value = "${digitalocean_droplet.web.ipv4_address}"
    type = "A"
}
```

Infrastructure as Code

- Human friendly config, JSON compatible
- Text format makes it version-able, VCS-friendly
- Declarative
- Infrastructure as code on a level not before possible

Zero to Done in One Command



Terraform Apply

```
$ terraform apply
digitalocean_droplet.web: Creating…
dnsimple_record.hello: Creating…

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Zero to Done in One Command

- Idempotent
- Highly parallelized
- Will only do what the plan says

Safely Change/Iterate

Terraform Plan

```
+ digitalocean_droplet.web
    backups:              "" => "<computed>"
    image:                "" => "centos-5-8-x32"
    ipv4_address:         "" => "<computed>"
    ipv4_address_private: "" => "<computed>"
    name:                 "" => "tf-web"
    private_networking:   "" => "<computed>"
    region:               "" => "sfo1"
    size:                 "" => "512mb"
    status:               "" => "<computed>"

+ dnsimple_record.hello
    domain:    "" => "example.com"
    domain_id: "" => "<computed>"
    hostname:  "" => "<computed>"
    name:      "" => "test"
    priority:  "" => "<computed>"
    ttl:       "" => "<computed>"
    type:      "" => "A"
    value:     "" => "${digitalocean_droplet.web.ipv4_address}"
```

Safely Change/Iterate

Terraform Plan

```
+ digitalocean_droplet.web
    backups:                 "" => "<computed>"
    image:                   "" => "centos-5-8-x32"
    ipv4_address:            "" => "<computed>"
    ipv4_address_private:    "" => "<computed>"
    name:                    "" => "tf-web"
    private_networking:      "" => "<computed>"
    region:                  "" => "sfo1"
    size:                    "" => "512mb"
    status:                  "" => "<computed>"

+ dnsimple_record.hello
    domain:     "" => "example.com"
    domain_id:  "" => "<computed>"
    hostname:   "" => "<computed>"
    name:       "" => "test"
    priority:   "" => "<computed>"
    ttl:        "" => "<computed>"
    type:       "" => "A"
    value:      "" => "${digitalocean_droplet.web.ipv4_address}"
```

Safely Change/Iterate

Terraform Plan

```
+ digitalocean_droplet.web
    backups:              "" => "<computed>"
    image:                "" => "centos-5-8-x32"
    ipv4_address:         "" => "<computed>"
    ipv4_address_private: "" => "<computed>"
    name:                 "" => "tf-web"
    private_networking:   "" => "<computed>"
    region:               "" => "sfo1"
    size:                 "" => "512mb"
    status:               "" => "<computed>"

+ dnsimple_record.hello
    domain:    "" => "example.com"
    domain_id: "" => "<computed>"
    hostname:  "" => "<computed>"
    name:      "" => "test"
    priority:  "" => "<computed>"
    ttl:       "" => "<computed>"
    type:      "" => "A"
    value:     "" => "${digitalocean_droplet.web.ipv4_address}"
```

Safely Change/Iterate

- Plan shows you what will happen
- Save plans to *guarantee* what will happen
- Plans show reasons for certain actions (such as re-create)
- Prior to Terraform: Operators had to “divine” change ordering, parallelization, rollout effect.

Lots more features…

- Modules for knowledge sharing, reusable components
- Remote state for resource sharing
- Targeted applies to limit effect of any change
- Lifecycle management
- Custom plugins are simple

Workflow

- Make code changes
- `terraform plan`
- Pull request with code changes + plan to make changes
- Review and merge
- `terraform apply pr1234.tfplan`

Terraform with Containers

Terraform with Docker

```
# Configure the Docker provider
provider "docker" {
    host = "tcp://127.0.0.1:1234/"
}

# Create a container
resource "docker_container" "foo" {
    image = "${docker_image.ubuntu.latest}"
    name = "foo"
}

resource "docker_image" "ubuntu" {
    name = "ubuntu:latest"
}
```

Terraform with Containers

Terraform with Docker

```
# Configure the Docker provider
provider "docker" {
    host = "tcp://127.0.0.1:1234/"
}

# Create a container
resource "docker_container" "foo" {
    image = "${docker_image.ubuntu.latest}"
    name = "foo"
}

resource "docker_image" "ubuntu" {
    name = "ubuntu:latest"
}
```

Terraform with Containers

Terraform with Docker

```
# Configure the Docker provider
provider "docker" {
    host = "tcp://127.0.0.1:1234/"
}

# Create a container
resource "docker_container" "foo" {
    image = "${docker_image.ubuntu.latest}"
    name = "foo"
}

resource "docker_image" "ubuntu" {
    name = "ubuntu:latest"
}
```

Terraform with Containers

Terraform with Docker

```
# Configure the Docker provider
provider "docker" {
    host = “tcp://127.0.0.1:1234/"
    alias = “foo”
}

# Create a container
resource "docker_container" "foo" {
    image = "${docker_image.ubuntu.latest}"
    name = “foo”
    provider = “docker.foo”
}
```

Terraform with Containers

Terraform with Docker

```
# Configure the Docker provider
provider "docker" {
    host = “tcp://127.0.0.1:1234/"
    alias = “foo”
}

# Create a container
resource "docker_container" "foo" {
    image = "${docker_image.ubuntu.latest}"
    name = “foo”
    provider = “docker.foo”
}
```

Terraform with Containers

Terraform with Docker

```
# Create a container
resource "docker_container" "foo" {
    image = "${docker_image.ubuntu.latest}"
    name = “foo”
    host = “tcp://127.0.0.1:1234/“
}
```

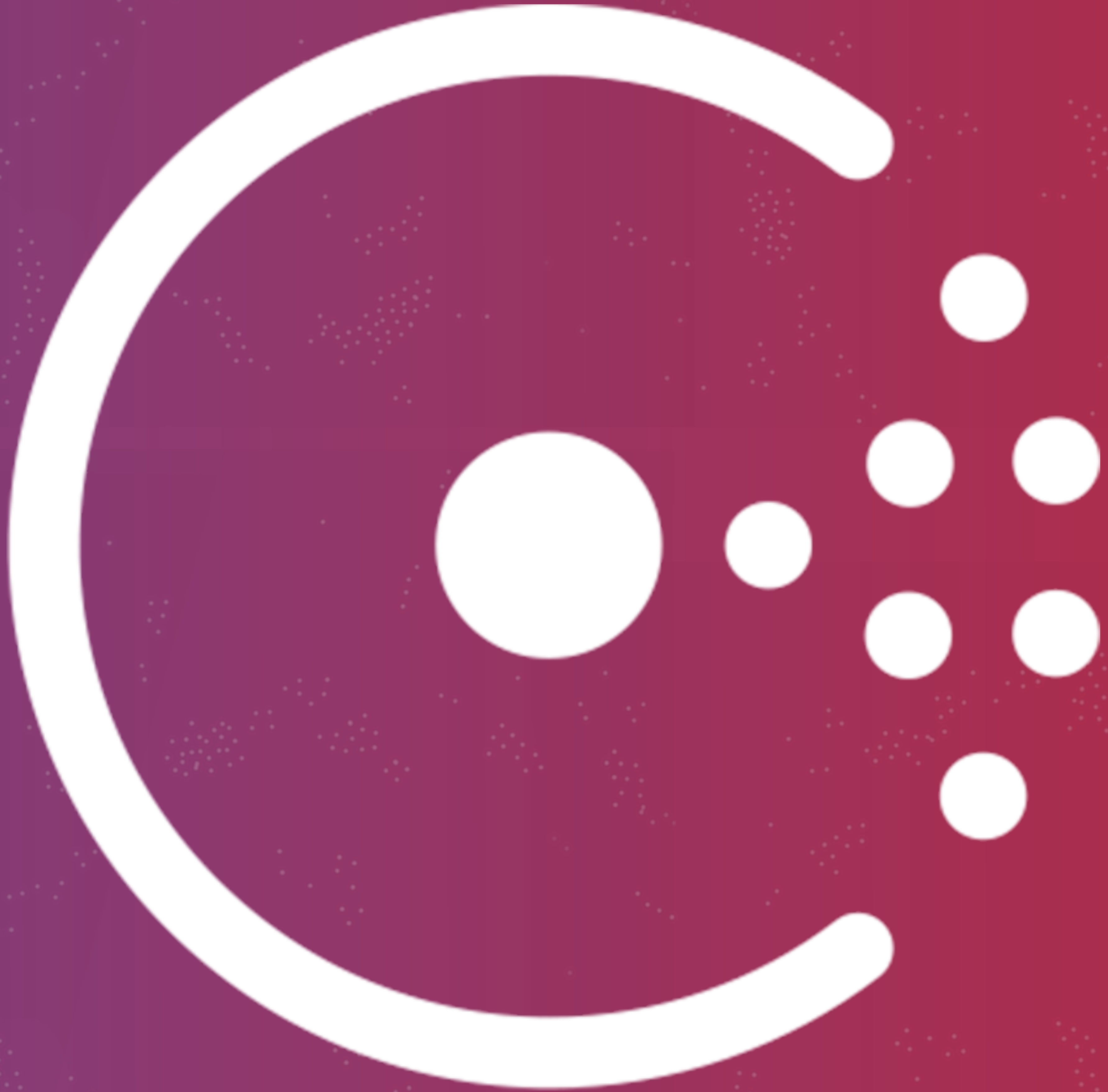
Terraform with Containers

Terraform with Docker

```
# Create a container
resource "docker_container" "foo" {
    image = "${docker_image.ubuntu.latest}"
    name = “foo”
    host = “tcp://127.0.0.1:1234/“
}
```

Terraform with Containers

- Manage both the underlying infrastructure *and* application-level containers
- Inherit lifecycle management features of Terraform
- Single host assign + schedulers



consul.io

Service discovery, configuration, and orchestration made easy. Distributed, highly available, and datacenter-aware.

Questions that Consul Answers

- Where is the service *foo*? (ex. Where is the database?)
- What is the health status of service *foo*?
- What is the health status of the machine/node *foo*?
- What is the list of all currently running machines?
- What is the configuration of service *foo*?
- Is anyone else currently performing operation *foo*?

Service Discovery

Where is service foo?

Service Discovery

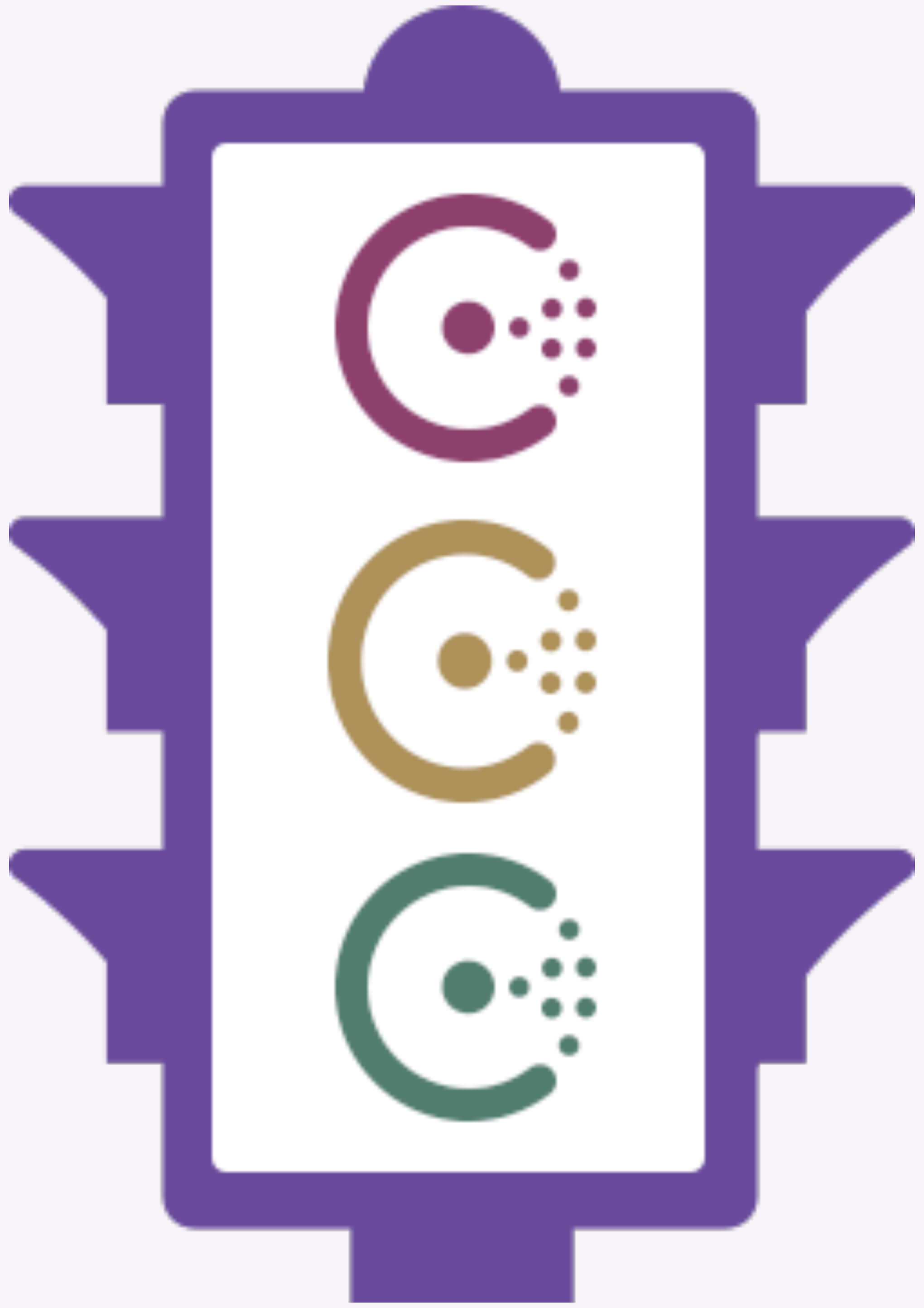
Service Discovery via DNS or HTTP

```
$ dig web-frontend.service.consul. +short
10.0.3.89
10.0.1.46

$ curl http://localhost:8500/v1/catalog/service/web-frontend
[{
  “Node”: “node-e818f1”,
  “Address”: “10.0.3.89”,
  “ServiceID”: “web-frontend”,
  …
}]
```

Service Discovery

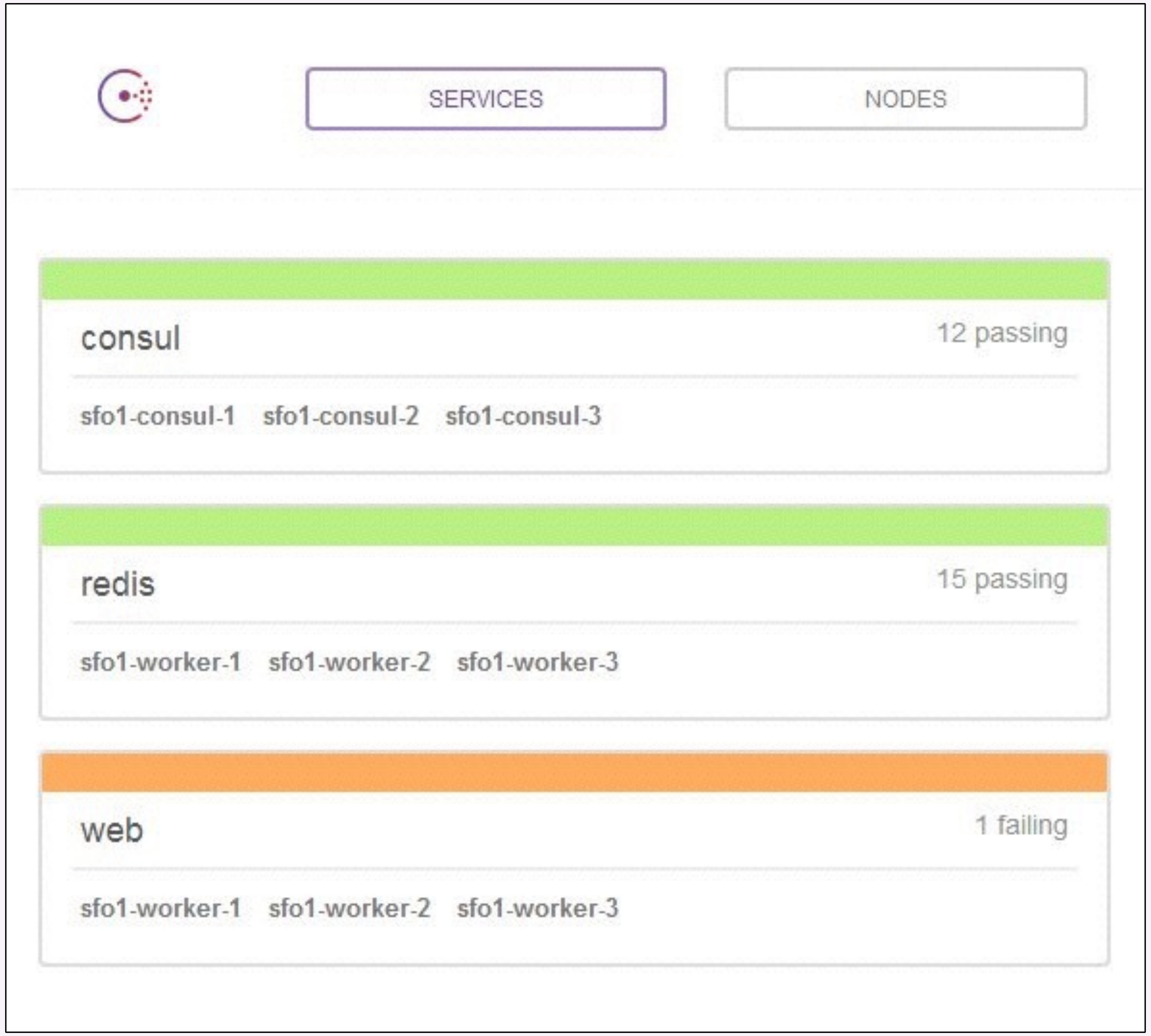
- DNS is legacy-friendly. No application changes required.
- HTTP returns rich metadata.
- Discover both internal and external services (such as service providers)



Failure Detection

Is service foo healthy/available?

Failure Detection



consul 12 passing

sfo1-consul-1 sfo1-consul-2 sfo1-consul-3

redis 15 passing

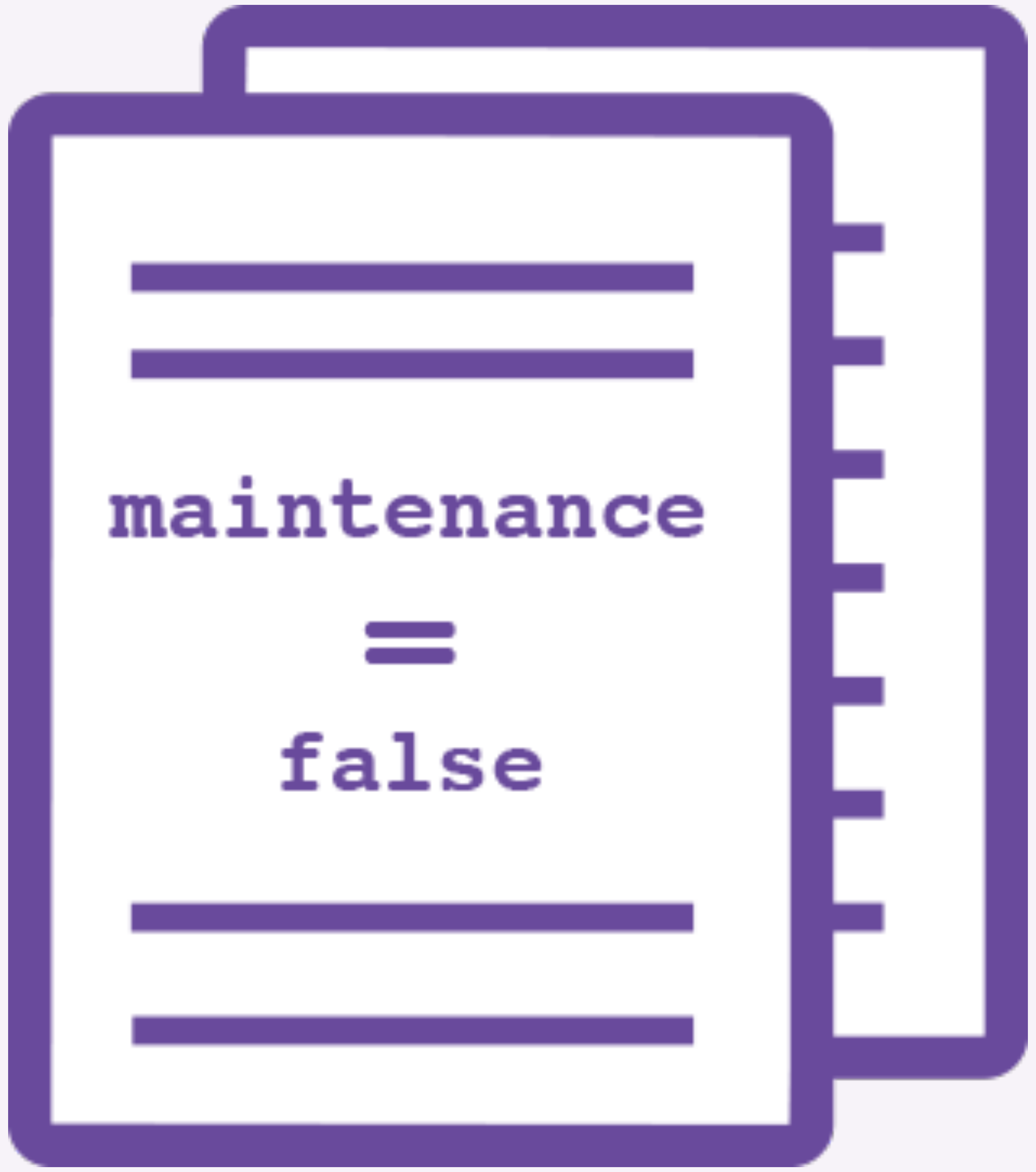
sfo1-worker-1 sfo1-worker-2 sfo1-worker-3

web 1 failing

sfo1-worker-1 sfo1-worker-2 sfo1-worker-3

Failure Detection

- DNS won’t return non-healthy services or nodes.
- HTTP has endpoints to list health state of catalog.



Key/Value Storage

What is the config of service foo?

Key/Value Storage

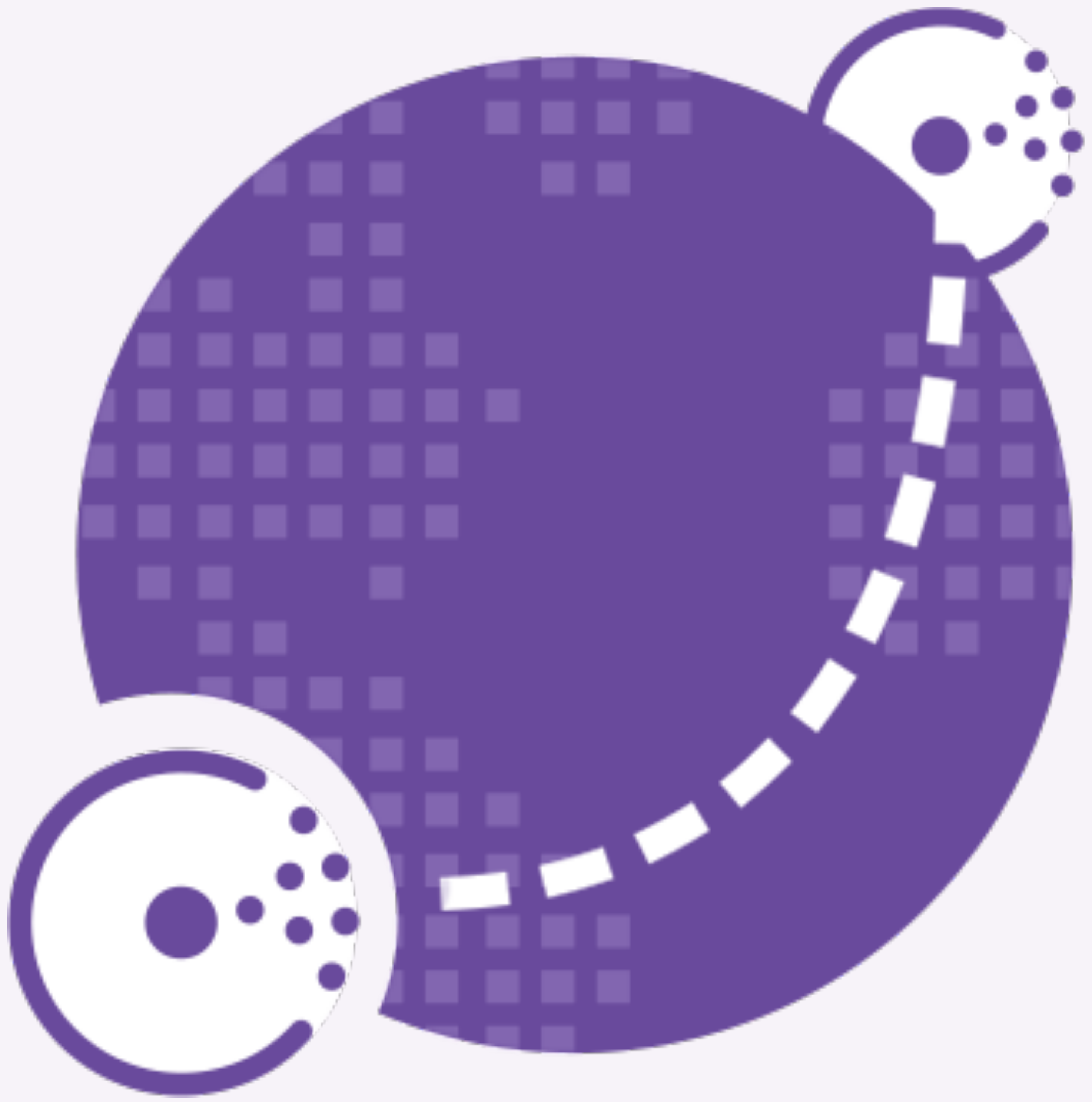
Setting and Getting a Key

```
$ curl -X PUT -d ‘bar’ http://localhost:8500/v1/kv/foo
true

$ curl http://localhost:8500/v1/kv/foo?raw
bar
```

Key/Value Storage

- Highly available storage of configuration.
- Turn knobs without big configuration management process.
- Watch keys (long poll) for changes
- ACLs on key/value to protect sensitive information



Multi-Datacenter

Multi-Datacenter

Service Discovery

```
$ dig web-frontend.singapore.service.consul. +short
10.3.3.33
10.3.1.18

$ dig web-frontend.germany.service.consul. +short
10.7.3.41
10.7.1.76
```

Multi-Datacenter

Setting and Getting a Key

```
$ curl http://localhost:8500/v1/kv/foo?raw&dc=asia
true

$ curl http://localhost:8500/v1/kv/foo?raw&dc=eu
false
```

Multi-Datacenter

- Local by default
- Can query other datacenters however you may need to
- Can view all datacenters within one UI

Orchestration

Events, Exec, Locks, Watches

Events, Exec, Watches

Dispatching Custom Events

```
$ consul event deploy 6DF7FE
…

$ consul watch -type event -name deploy /usr/bin/deploy.sh
…

$ consul exec -service web /usr/bin/deploy.sh
…
```

Events, Exec, Watches

- Powerful orchestration tools
- Pros/cons to each approach, use the right tool for the job
- All approaches proven to scale to thousands of agents

Locks

Dispatching Custom Events

```
$ consul lock ./deploy.sh
…

$ consul lock -n=5 ./deploy.sh
…
```

Locks

- Distributed lock
- Can have up to *n* acquisitions (semaphore)
- Primitive to make redundant but serial services

Operational Bullet Points

- Leader election via Raft
- Gossip protocol for aliveness
- Three consistency models: default, consistent, and stale
- Encryption, ACLs available
- Real world usage to thousands of agents per datacenter

Consul with Containers

- Run in or outside the container
- Runtime configuration vs. buildtime configuration
- Discover non-container services, plus transparent change if/when they become containers
- Speed and scalability of Consul very easily ready for “container scale” as well as future scale

Thanks!

QUESTIONS?