Operating the Open Core

Ryan Uber

@ryanuber

CISCO

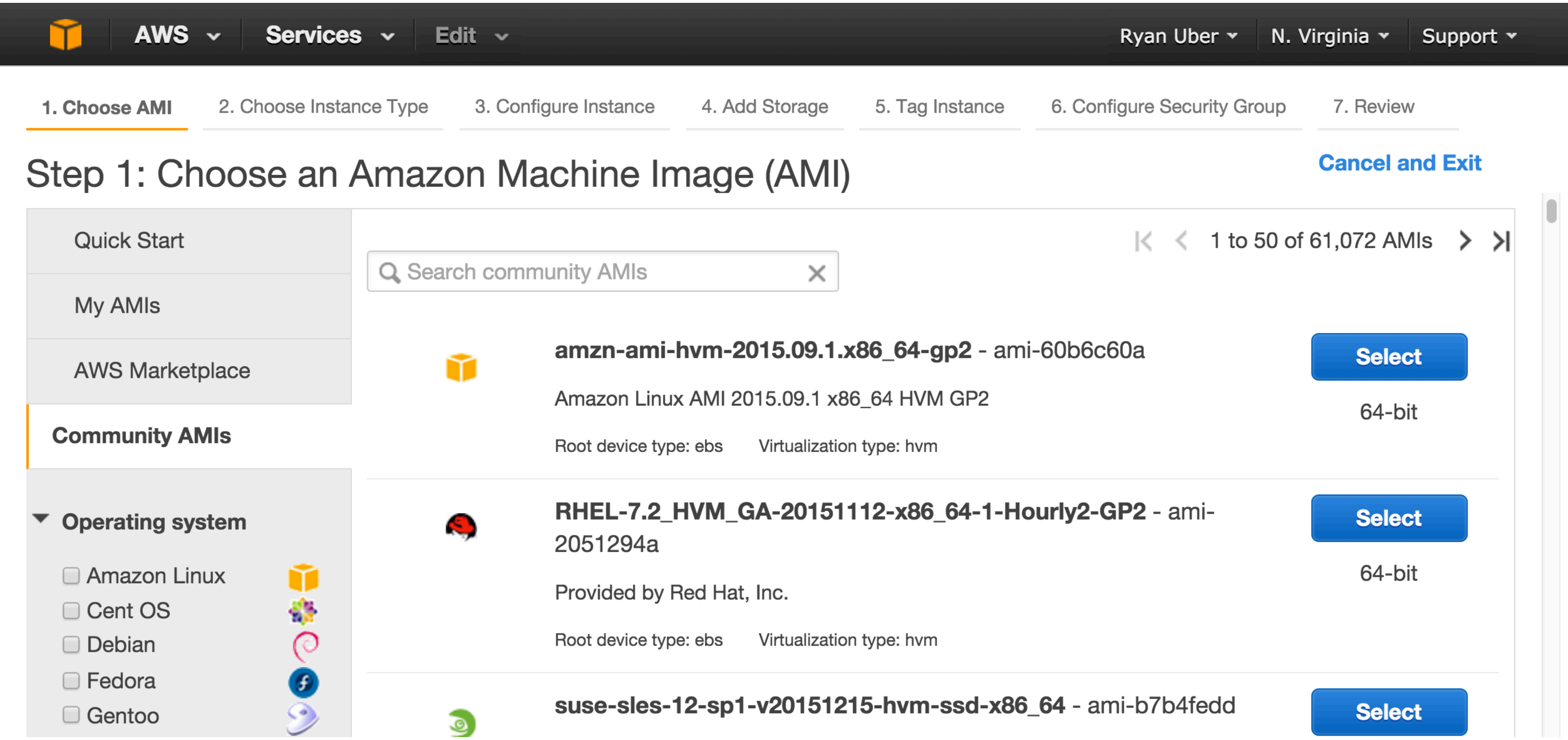
Packer

https://packer.io

What is Packer?

- Create machine and container images
- For multiple platforms
- From a single source configuration

- Create machine and container images
- For multiple platforms
- From a single source configuration



Step 1: Choose an Amazon Machine Image (AMI)

1 to 50 of 61,072 AMIs

Operating system

- Amazon Linux
- Cent OS
- Debian
- Fedora
- Gentoo

amzn-ami-hvm-2015.09.1.x86_64-gp2 - ami-60b6c60a

Amazon Linux AMI 2015.09.1 x86_64 HVM GP2

Root device type: ebs Virtualization type: hvm

64-bit

RHEL-7.2_HVM_GA-20151112-x86_64-1-Hourly2-GP2 - ami-2051294a

Provided by Red Hat, Inc.

Root device type: ebs Virtualization type: hvm

64-bit

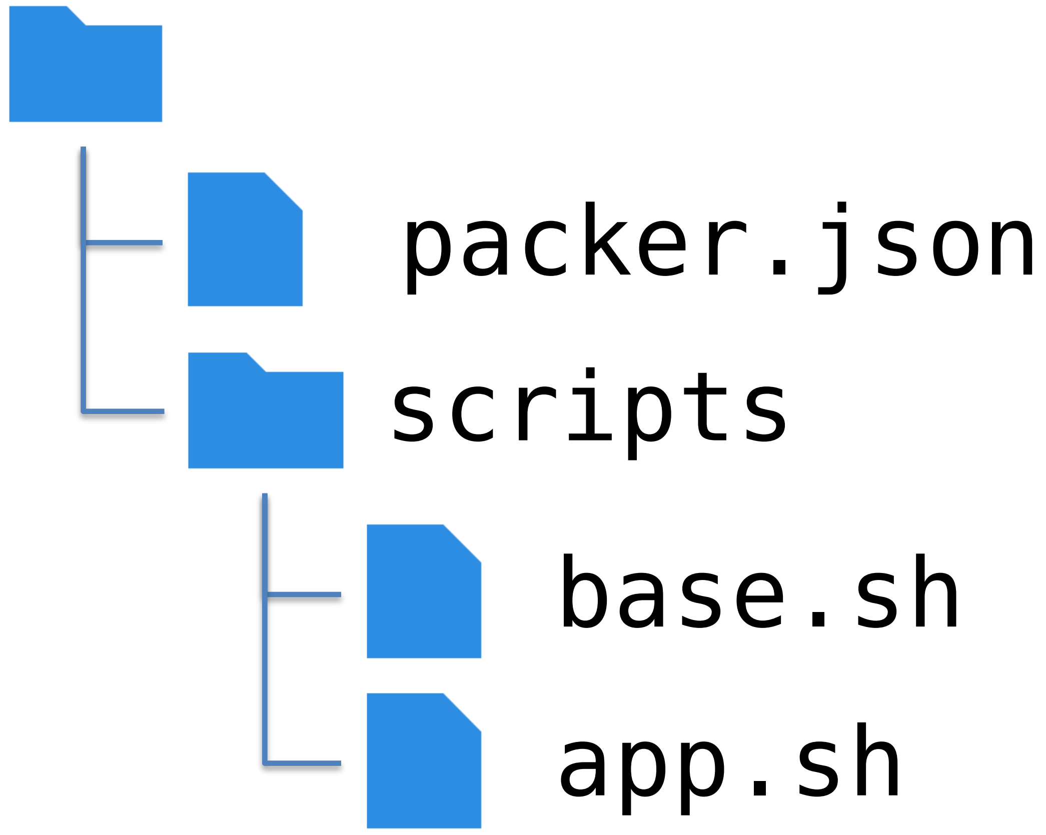
suse-sles-12-sp1-v20151215-hvm-ssd-x86_64 - ami-b7b4fedd

Reproducible?

Maintainable?

Automatic?

Encapsulation



Provisioners

- Basic shell scripts
- Puppet
- Chef
- File uploads
- Many more…

Uniformity

```
> packer build ./packer.json
```

Predictability

- Create machine and container images
- For multiple platforms
- From a single source configuration

Build these all separately?

...

- Log in to platform
- Create and start an instance
- SSH to instance
- Copy scripts / binaries
- Run commands
- Shutdown
- Snapshot

- Log in to platform
- Create and start an instance
- SSH to instance
- Copy scripts / binaries
- Run commands
- Shutdown
- Snapshot

… FOR EVERY PLATFORM!?

- ~~Log in to platform~~
- ~~Create and start an instance~~
- ~~SSH to instance~~
- ~~Copy scripts / binaries~~
- ~~Run commands~~
- ~~Shutdown~~
- ~~Snapshot~~

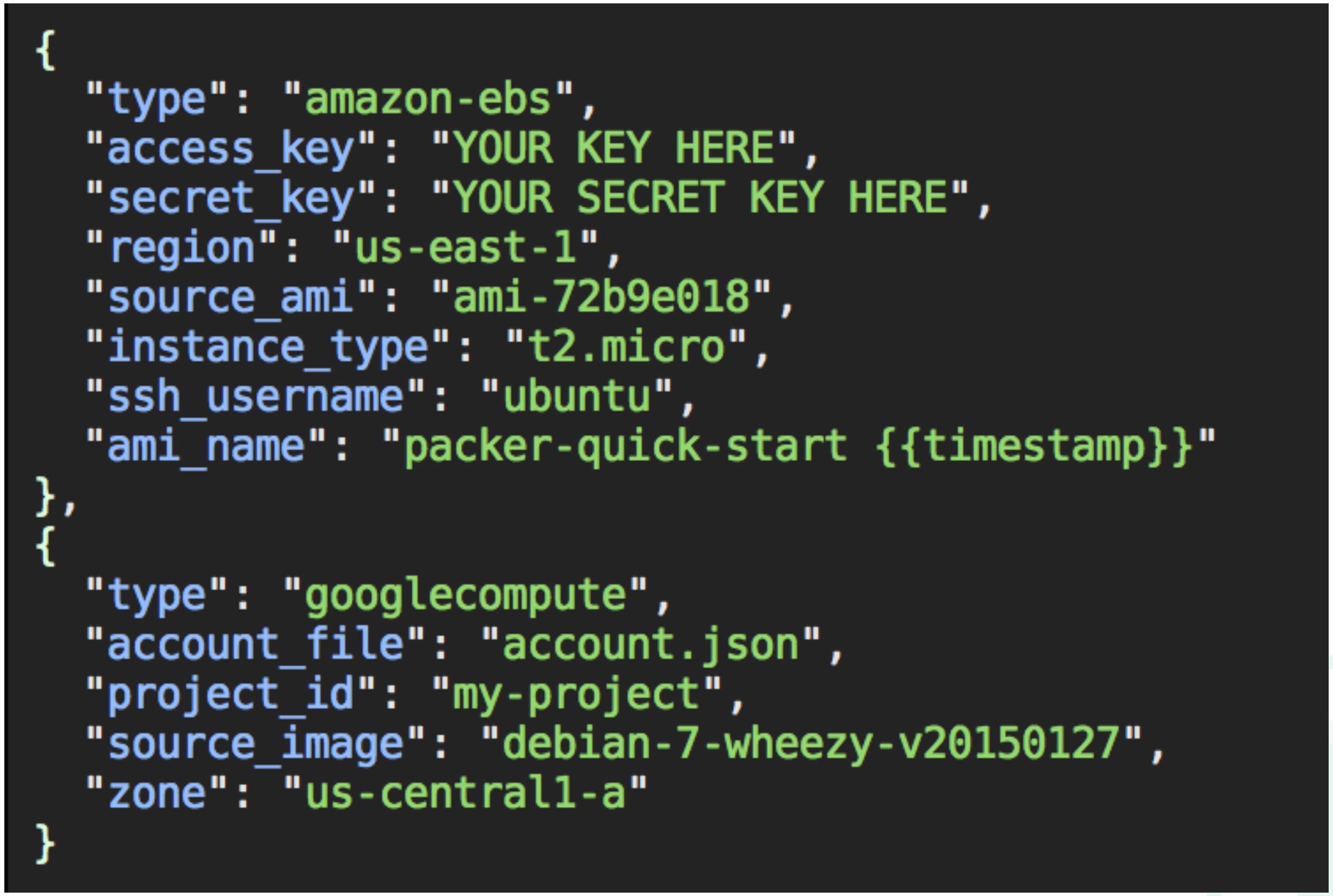
~~… FOR EVERY PLATFORM!?~~

Builders

1. Expose platform-specific setup instructions

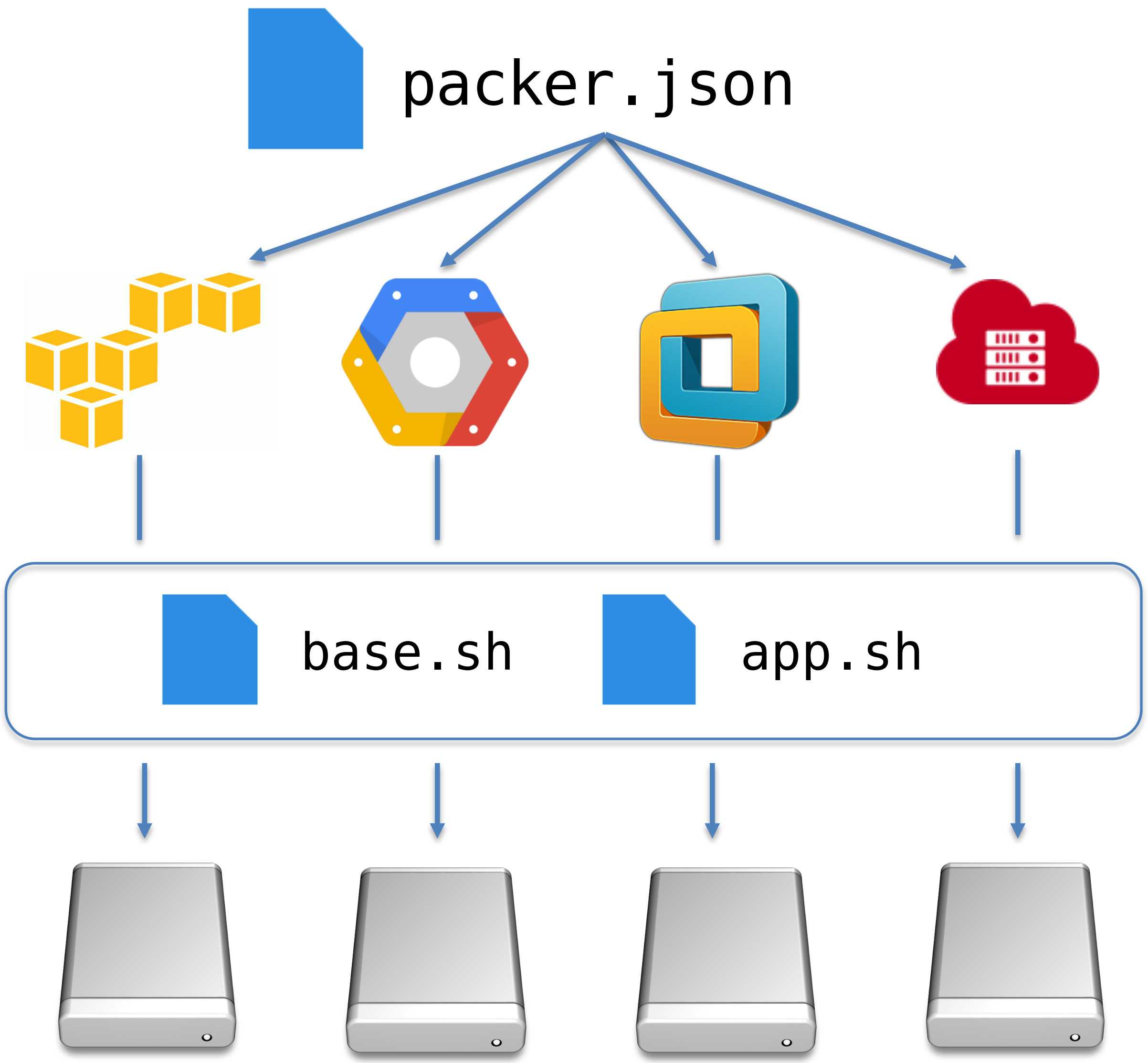
2. Provide a common hand-off to provisioning scripts

Builders



```
{
  "type": "amazon-ebs",
  "access_key": "YOUR KEY HERE",
  "secret_key": "YOUR SECRET KEY HERE",
  "region": "us-east-1",
  "source_ami": "ami-72b9e018",
  "instance_type": "t2.micro",
  "ssh_username": "ubuntu",
  "ami_name": "packer-quick-start {{timestamp}}"
},
{
  "type": "googlecompute",
  "account_file": "account.json",
  "project_id": "my-project",
  "source_image": "debian-7-wheezy-v20150127",
  "zone": "us-central1-a"
}
```

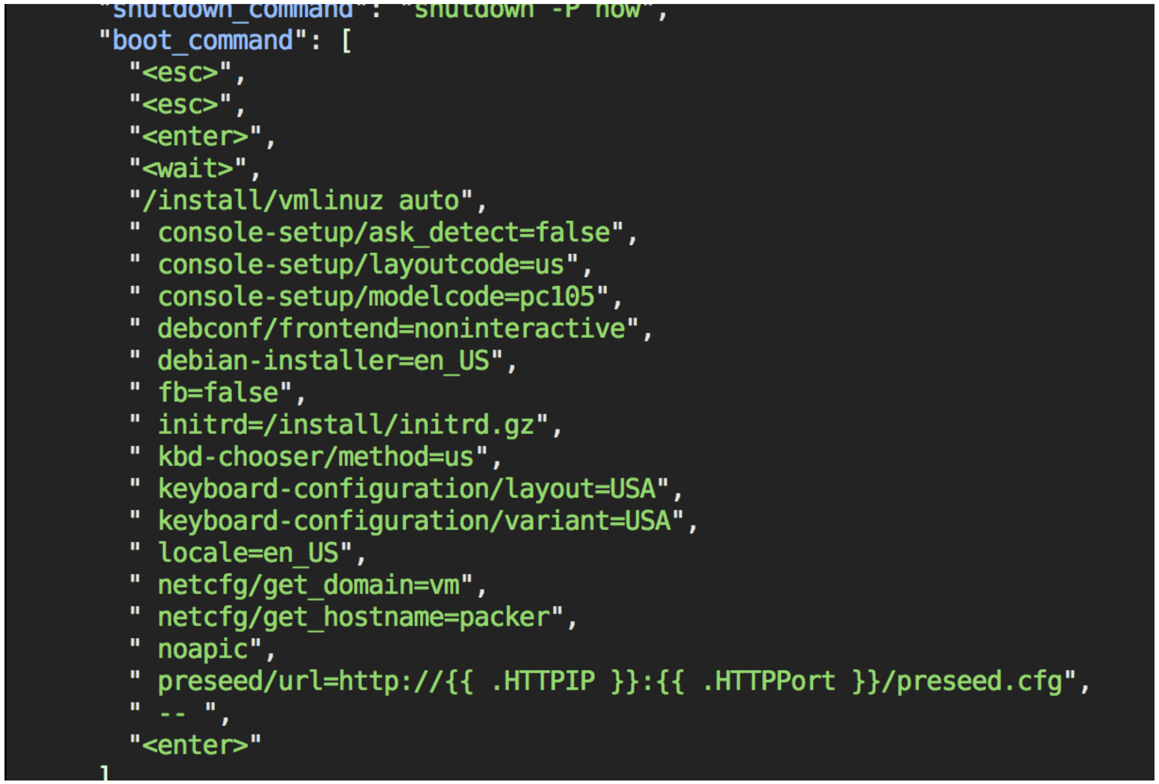
- Create machine and container images
- For multiple platforms
- From a single source configuration


packer.json
base.sh
app.sh

From the ground up

VMware
Virtualbox
QEMU / KVM
Parallels

Automated ISO installs



```
    "shutdown_command": "shutdown -P now",
    "boot_command": [
      "<esc>",
      "<esc>",
      "<enter>",
      "<wait>",
      "/install/vmlinuz auto",
      " console-setup/ask_detect=false",
      " console-setup/layoutcode=us",
      " console-setup/modelcode=pc105",
      " debconf/frontend=noninteractive",
      " debian-installer=en_US",
      " fb=false",
      " initrd=/install/initrd.gz",
      " kbd-chooser/method=us",
      " keyboard-configuration/layout=USA",
      " keyboard-configuration/variant=USA",
      " locale=en_US",
      " netcfg/get_domain=vm",
      " netcfg/get_hostname=packer",
      " noapic",
      " preseed/url=http://{{ .HTTPIP }}:{{ .HTTPPort }}/preseed.cfg",
      " -- ",
      "<enter>"
    ]
```

Post Processors

Vagrant	Convert to Vagrant .box format
Atlas	Publish to HashiCorp Atlas
Docker	Save locally, publish to hub, etc.

How HashiCorp uses Packer

Building Images for Production Services

- Modify base operating system installation (“Masterless” puppet single-apply)
- Install pre-compiled applications
- Prepare service discovery (Consul)
- Result is an “immutable” image

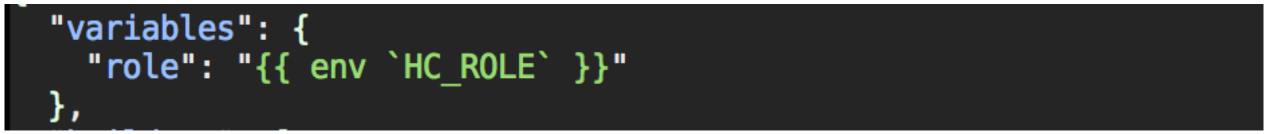
Building Images for Production Services

Only one Packer template (for everything)

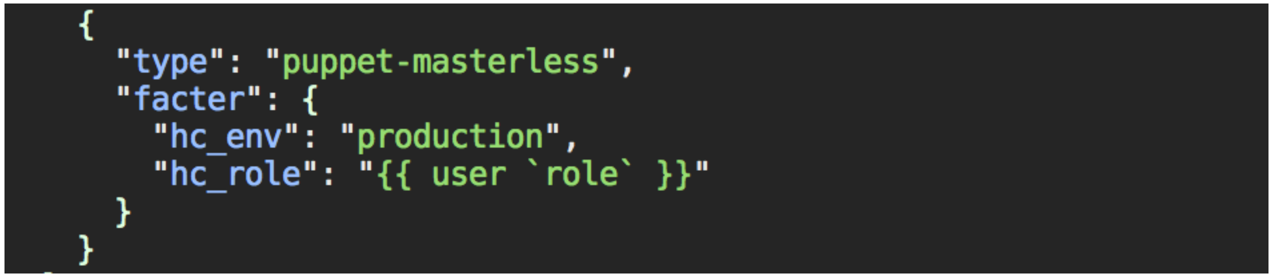
```
> HC_ROLE=binstore packer build packer.json
```

Building Images for Production Services

Packer

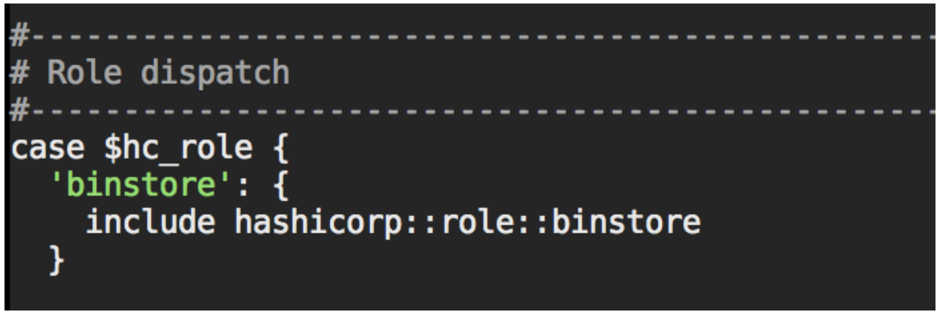


```
  "variables": {
    "role": "{{ env `HC_ROLE` }}"
  },
```



```
    {
      "type": "puppet-masterless",
      "facter": {
        "hc_env": "production",
        "hc_role": "{{ user `role` }}"
      }
    }
```

Puppet



```
#------------------------------------------
# Role dispatch
#------------------------------------------
case $hc_role {
  'binstore': {
    include hashicorp::role::binstore
  }
```

Building VMware machines for isolation

- Typical Packer template, VMware provider
- Prepares a “base” disk image, base OS only
- Disk image cloned for each unit of work
- VMware for nested virtualization

Application Compilation

- On-demand builds for any application
- Docker for speed and runtime availability
- Post-processors for artifact extraction

Maintaining Vagrant Boxes

- Easy for multiple platforms and architectures
- Post-processor for Vagrant-specific setup

Packer Questions?

Terraform

https://terraform.io

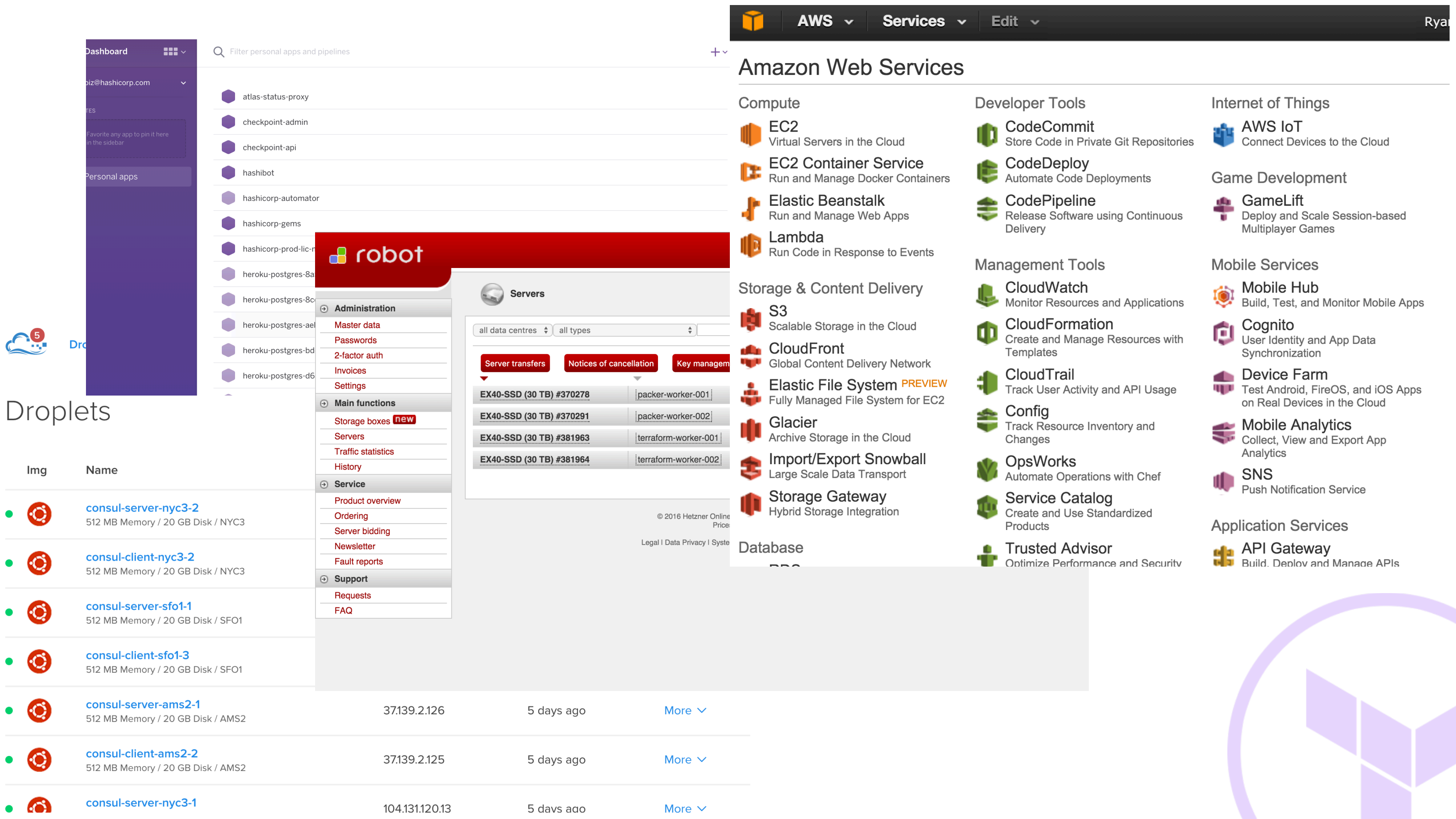
What is Terraform?

Terraform is a tool to execute infrastructure operations:

- Build
- Combine
- Launch

- Build
- Combine
- Launch

How do I deploy my app?



Dashboard

biz@hashicorp.com

Favorite any app to pin it here in the sidebar

Personal apps

Filter personal apps and pipelines

atlas-status-proxy
checkpoint-admin
checkpoint-api
hashibot
hashicorp-automator
hashicorp-gems
hashicorp-prod-lic-r
heroku-postgres-8a
heroku-postgres-8c
heroku-postgres-ael
heroku-postgres-bd
heroku-postgres-d6

robot

Administration
Master data
Passwords
2-factor auth
Invoices
Settings

Main functions
Storage boxes new
Servers
Traffic statistics
History

Service
Product overview
Ordering
Server bidding
Newsletter
Fault reports

Support
Requests
FAQ

Servers

all data centres | all types

Server transfers | Notices of cancellation | Key managem

EX40-SSD (30 TB) #370278	packer-worker-001
EX40-SSD (30 TB) #370291	packer-worker-002
EX40-SSD (30 TB) #381963	terraform-worker-001
EX40-SSD (30 TB) #381964	terraform-worker-002

© 2016 Hetzner Online
Legal | Data Privacy | Syste

AWS | Services | Edit | Rya

Amazon Web Services

Compute
EC2 — Virtual Servers in the Cloud
EC2 Container Service — Run and Manage Docker Containers
Elastic Beanstalk — Run and Manage Web Apps
Lambda — Run Code in Response to Events

Storage & Content Delivery
S3 — Scalable Storage in the Cloud
CloudFront — Global Content Delivery Network
Elastic File System PREVIEW — Fully Managed File System for EC2
Glacier — Archive Storage in the Cloud
Import/Export Snowball — Large Scale Data Transport
Storage Gateway — Hybrid Storage Integration

Database

Developer Tools
CodeCommit — Store Code in Private Git Repositories
CodeDeploy — Automate Code Deployments
CodePipeline — Release Software using Continuous Delivery

Management Tools
CloudWatch — Monitor Resources and Applications
CloudFormation — Create and Manage Resources with Templates
CloudTrail — Track User Activity and API Usage
Config — Track Resource Inventory and Changes
OpsWorks — Automate Operations with Chef
Service Catalog — Create and Use Standardized Products
Trusted Advisor — Optimize Performance and Security

Internet of Things
AWS IoT — Connect Devices to the Cloud

Game Development
GameLift — Deploy and Scale Session-based Multiplayer Games

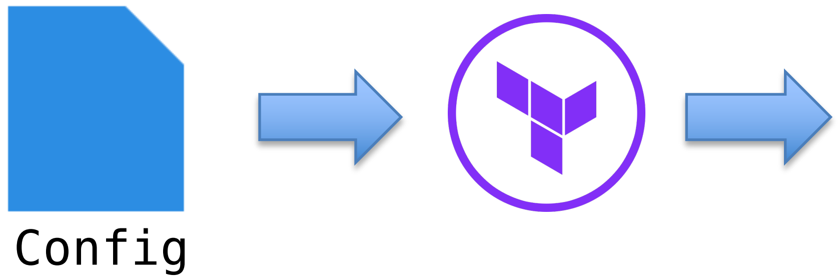
Mobile Services
Mobile Hub — Build, Test, and Monitor Mobile Apps
Cognito — User Identity and App Data Synchronization
Device Farm — Test Android, FireOS, and iOS Apps on Real Devices in the Cloud
Mobile Analytics — Collect, View and Export App Analytics
SNS — Push Notification Service

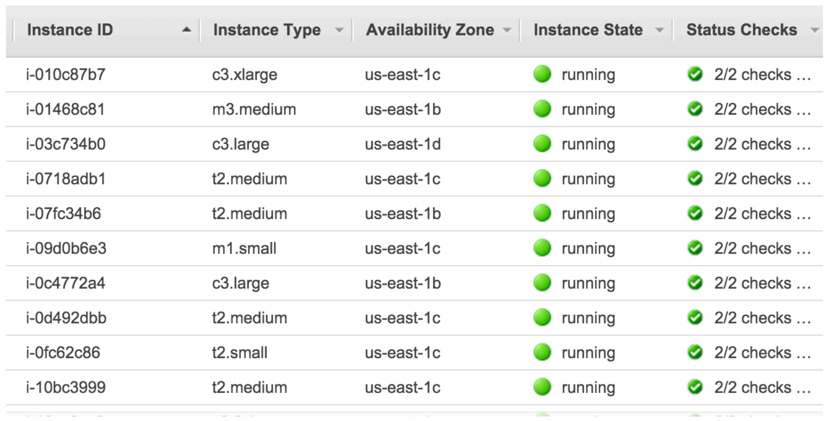
Application Services
API Gateway — Build, Deploy and Manage APIs

Droplets

Img	Name			
●	consul-server-nyc3-2 512 MB Memory / 20 GB Disk / NYC3			
●	consul-client-nyc3-2 512 MB Memory / 20 GB Disk / NYC3			
●	consul-server-sfo1-1 512 MB Memory / 20 GB Disk / SFO1			
●	consul-client-sfo1-3 512 MB Memory / 20 GB Disk / SFO1			
●	consul-server-ams2-1 512 MB Memory / 20 GB Disk / AMS2	37.139.2.126	5 days ago	More
●	consul-client-ams2-2 512 MB Memory / 20 GB Disk / AMS2	37.139.2.125	5 days ago	More
●	consul-server-nyc3-1	104.131.120.13	5 days ago	More

Deployment as an FSM





Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
i-010c87b7	c3.xlarge	us-east-1c	● running	✓ 2/2 checks ...
i-01468c81	m3.medium	us-east-1b	● running	✓ 2/2 checks ...
i-03c734b0	c3.large	us-east-1d	● running	✓ 2/2 checks ...
i-0718adb1	t2.medium	us-east-1c	● running	✓ 2/2 checks ...
i-07fc34b6	t2.medium	us-east-1b	● running	✓ 2/2 checks ...
i-09d0b6e3	m1.small	us-east-1c	● running	✓ 2/2 checks ...
i-0c4772a4	c3.large	us-east-1b	● running	✓ 2/2 checks ...
i-0d492dbb	t2.medium	us-east-1c	● running	✓ 2/2 checks ...
i-0fc62c86	t2.small	us-east-1c	● running	✓ 2/2 checks ...
i-10bc3999	t2.medium	us-east-1c	● running	✓ 2/2 checks ...

Terraform Internals

- No server component (CLI only)
- Human-/machine-readable config
- Graph-based (DAG)
- Pluggable providers

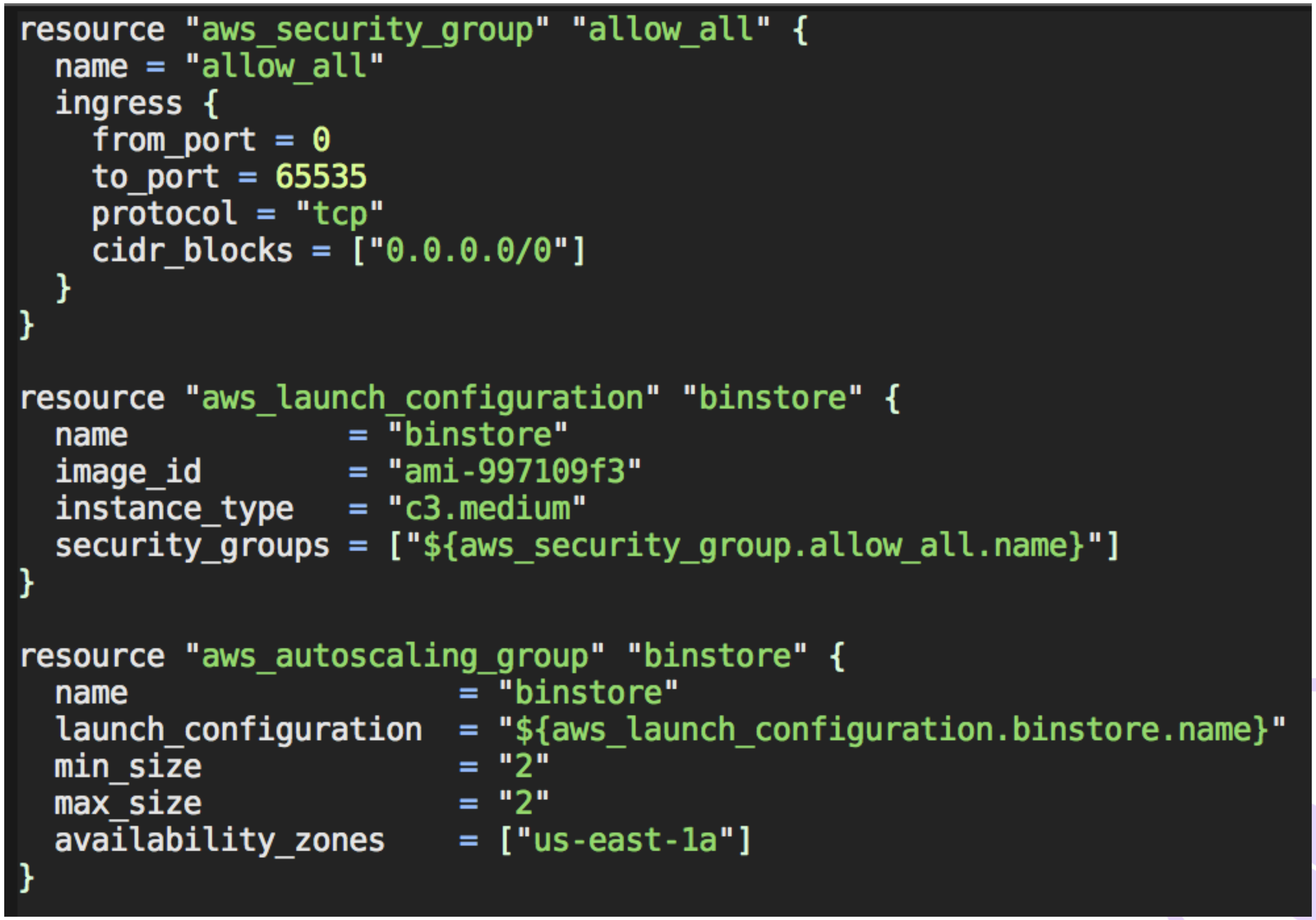
Terraform Workflow

- Write or make changes to infrastructure configuration
  - Deploy new service
  - Scale up existing service
  - Add new DNS records
  - Create databases

- Generate a plan: What steps to realize the changes?
  - Add/remove instances
  - Create DNS records
  - Create databases

- Apply the plan to mutate infrastructure

Step 1: Configuration

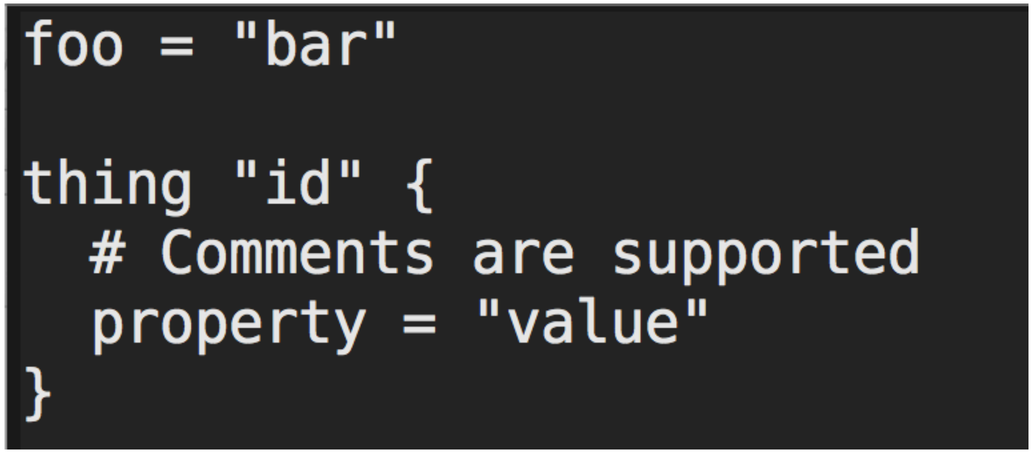


```
resource "aws_security_group" "allow_all" {
  name = "allow_all"
  ingress {
    from_port = 0
    to_port = 65535
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_launch_configuration" "binstore" {
  name            = "binstore"
  image_id        = "ami-997109f3"
  instance_type   = "c3.medium"
  security_groups = ["${aws_security_group.allow_all.name}"]
}

resource "aws_autoscaling_group" "binstore" {
  name                 = "binstore"
  launch_configuration = "${aws_launch_configuration.binstore.name}"
  min_size             = "2"
  max_size             = "2"
  availability_zones   = ["us-east-1a"]
}
```

HCL

https://github.com/hashicorp/hcl

Similar to libucl, nginx

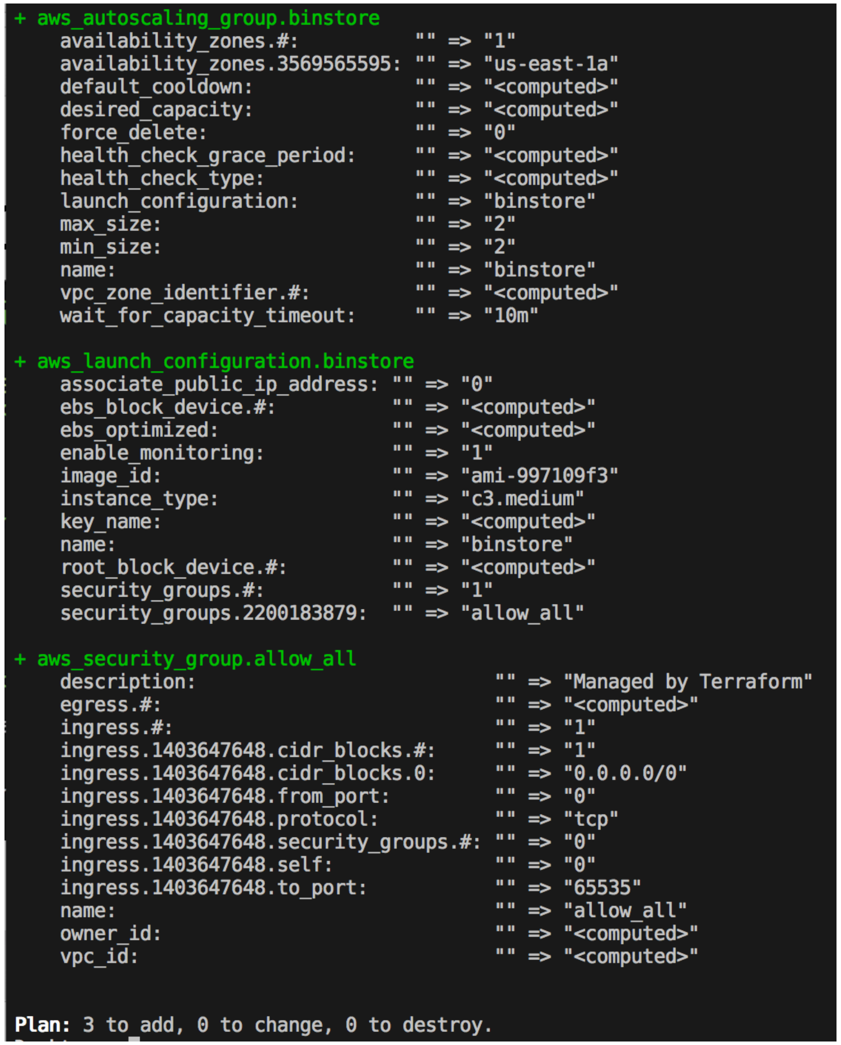


```
foo = "bar"

thing "id" {
  # Comments are supported
  property = "value"
}
```

Step 2: Plan

```
> terraform plan
```

Step 2: Plan

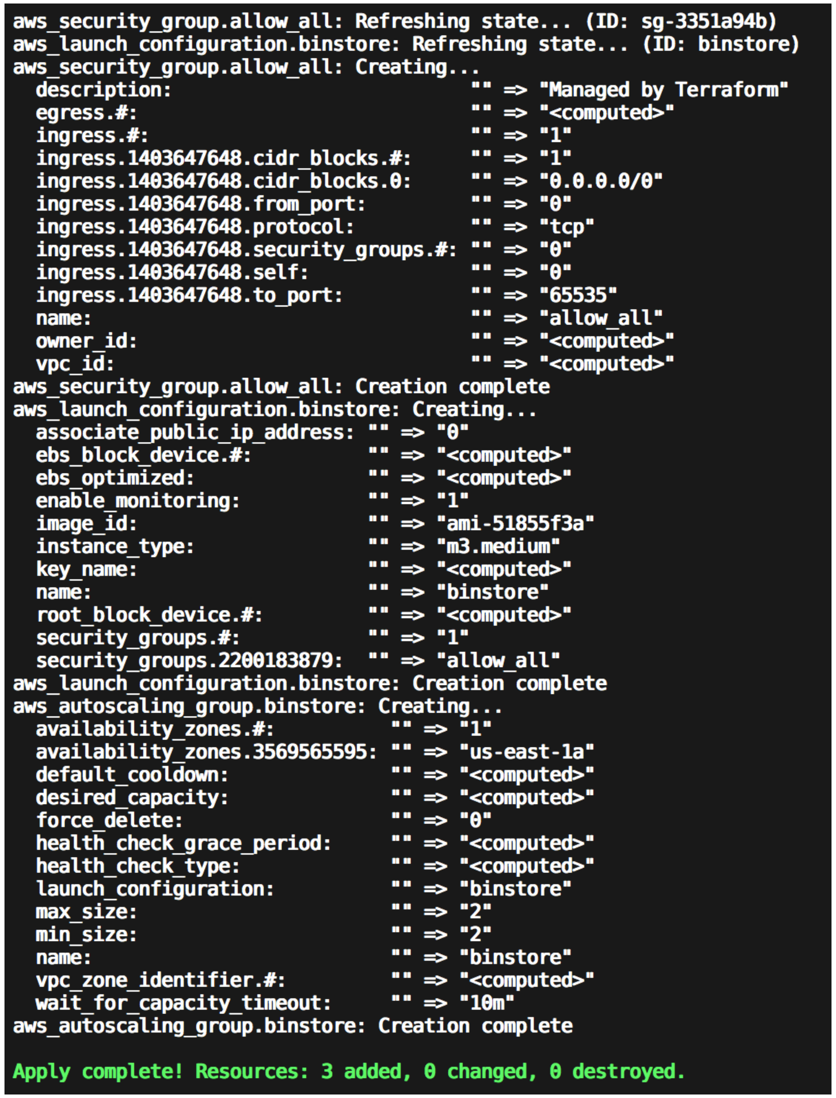


```
+ aws_autoscaling_group.binstore
    availability_zones.#:          "" => "1"
    availability_zones.3569565595: "" => "us-east-1a"
    default_cooldown:              "" => "<computed>"
    desired_capacity:              "" => "<computed>"
    force_delete:                  "" => "0"
    health_check_grace_period:     "" => "<computed>"
    health_check_type:             "" => "<computed>"
    launch_configuration:          "" => "binstore"
    max_size:                      "" => "2"
    min_size:                      "" => "2"
    name:                          "" => "binstore"
    vpc_zone_identifier.#:         "" => "<computed>"
    wait_for_capacity_timeout:     "" => "10m"

+ aws_launch_configuration.binstore
    associate_public_ip_address: "" => "0"
    ebs_block_device.#:          "" => "<computed>"
    ebs_optimized:               "" => "<computed>"
    enable_monitoring:           "" => "1"
    image_id:                    "" => "ami-997109f3"
    instance_type:               "" => "c3.medium"
    key_name:                    "" => "<computed>"
    name:                        "" => "binstore"
    root_block_device.#:         "" => "<computed>"
    security_groups.#:           "" => "1"
    security_groups.2200183879:  "" => "allow_all"

+ aws_security_group.allow_all
    description:                            "" => "Managed by Terraform"
    egress.#:                               "" => "<computed>"
    ingress.#:                              "" => "1"
    ingress.1403647648.cidr_blocks.#:       "" => "1"
    ingress.1403647648.cidr_blocks.0:       "" => "0.0.0.0/0"
    ingress.1403647648.from_port:           "" => "0"
    ingress.1403647648.protocol:            "" => "tcp"
    ingress.1403647648.security_groups.#:   "" => "0"
    ingress.1403647648.self:                "" => "0"
    ingress.1403647648.to_port:             "" => "65535"
    name:                                   "" => "allow_all"
    owner_id:                               "" => "<computed>"
    vpc_id:                                 "" => "<computed>"


Plan: 3 to add, 0 to change, 0 to destroy.
```

Step 3: Apply

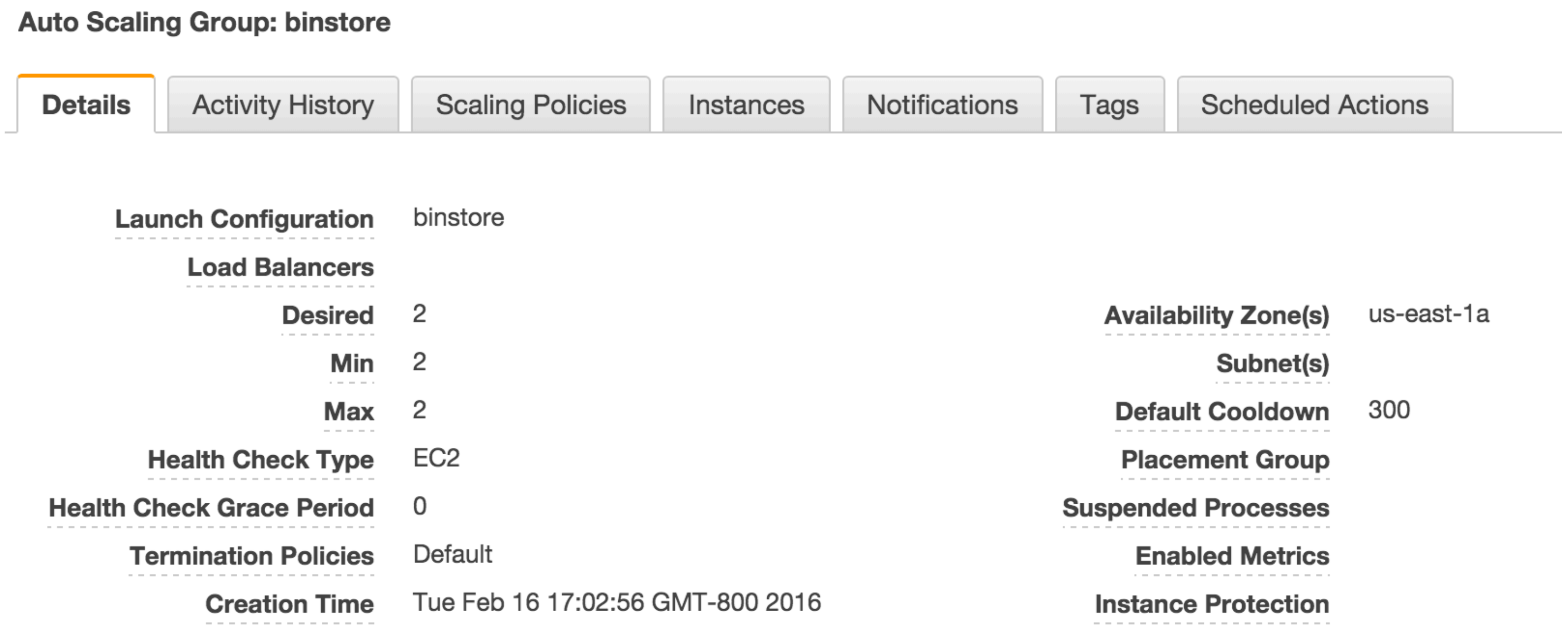
```
> terraform apply
```

Step 3: Apply



```
aws_security_group.allow_all: Refreshing state... (ID: sg-3351a94b)
aws_launch_configuration.binstore: Refreshing state... (ID: binstore)
aws_security_group.allow_all: Creating...
  description:                            "" => "Managed by Terraform"
  egress.#:                               "" => "<computed>"
  ingress.#:                              "" => "1"
  ingress.1403647648.cidr_blocks.#:       "" => "1"
  ingress.1403647648.cidr_blocks.0:       "" => "0.0.0.0/0"
  ingress.1403647648.from_port:           "" => "0"
  ingress.1403647648.protocol:            "" => "tcp"
  ingress.1403647648.security_groups.#:  "" => "0"
  ingress.1403647648.self:                "" => "0"
  ingress.1403647648.to_port:             "" => "65535"
  name:                                   "" => "allow_all"
  owner_id:                               "" => "<computed>"
  vpc_id:                                 "" => "<computed>"
aws_security_group.allow_all: Creation complete
aws_launch_configuration.binstore: Creating...
  associate_public_ip_address:  "" => "0"
  ebs_block_device.#:           "" => "<computed>"
  ebs_optimized:                "" => "<computed>"
  enable_monitoring:            "" => "1"
  image_id:                     "" => "ami-51855f3a"
  instance_type:                "" => "m3.medium"
  key_name:                     "" => "<computed>"
  name:                         "" => "binstore"
  root_block_device.#:          "" => "<computed>"
  security_groups.#:            "" => "1"
  security_groups.2200183879:   "" => "allow_all"
aws_launch_configuration.binstore: Creation complete
aws_autoscaling_group.binstore: Creating...
  availability_zones.#:           "" => "1"
  availability_zones.3569565595:  "" => "us-east-1a"
  default_cooldown:               "" => "<computed>"
  desired_capacity:               "" => "<computed>"
  force_delete:                   "" => "0"
  health_check_grace_period:      "" => "<computed>"
  health_check_type:              "" => "<computed>"
  launch_configuration:           "" => "binstore"
  max_size:                       "" => "2"
  min_size:                       "" => "2"
  name:                           "" => "binstore"
  vpc_zone_identifier.#:          "" => "<computed>"
  wait_for_capacity_timeout:      "" => "10m"
aws_autoscaling_group.binstore: Creation complete

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

Step 3: Apply

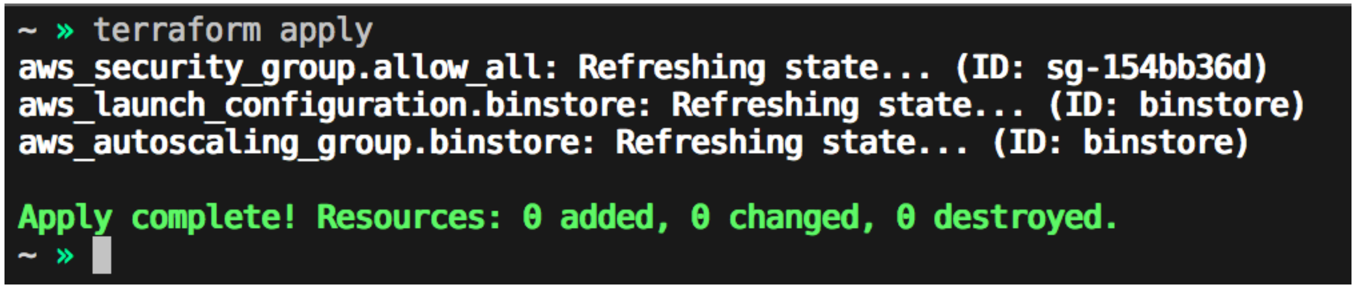


Auto Scaling Group: binstore

Launch Configuration	binstore		
Load Balancers			
Desired	2	**Availability Zone(s)**	us-east-1a
Min	2	**Subnet(s)**	
Max	2	**Default Cooldown**	300
Health Check Type	EC2	**Placement Group**	
Health Check Grace Period	0	**Suspended Processes**	
Termination Policies	Default	**Enabled Metrics**	
Creation Time	Tue Feb 16 17:02:56 GMT-800 2016	**Instance Protection**	

Step 3: Apply

Apply is idempotent



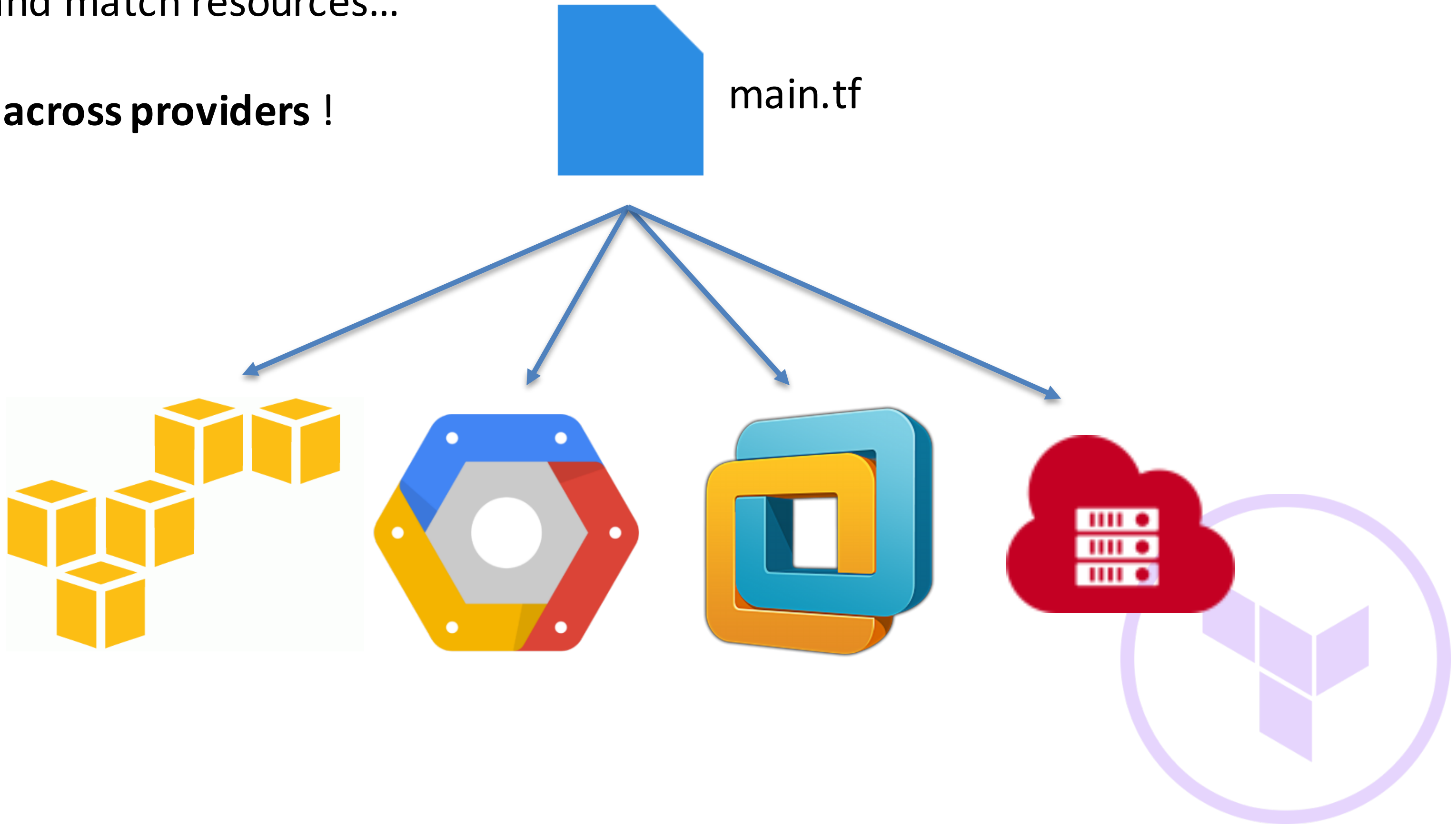
```
~ » terraform apply
aws_security_group.allow_all: Refreshing state... (ID: sg-154bb36d)
aws_launch_configuration.binstore: Refreshing state... (ID: binstore)
aws_autoscaling_group.binstore: Refreshing state... (ID: binstore)

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

- Build
- Combine
- Launch

No Provider Lock-in

Mix and match resources…

Even **across providers** !



Combining Providers

- Use differentiating resources from numerous providers to get best-of-the-bunch
- Fill in functionality gaps
- Makes infrastructure flexible

Code Reuse

- Infrastructure code can be very repetitive
- Separate environments effectively multiply the SLOC
- Copy/pasting code is error-prone and a maintenance nightmare

How does Terraform address this?

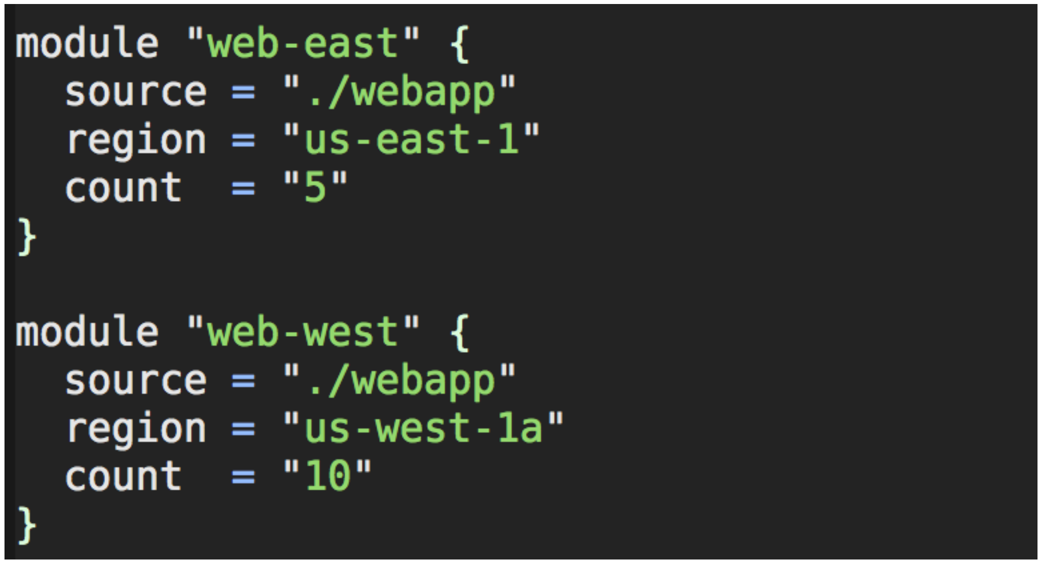
Modules

Essentially directories of Terraform configuration files

```
main.tf
webapp/
├─ main.tf
└─ variables.tf
redis/
├─ main.tf
└─ variables.tf
```

Modules

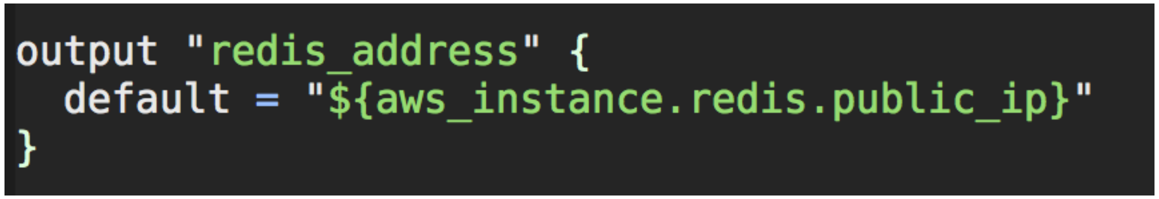
Callable and paramaterizable, similar to functions



```
module "web-east" {
  source = "./webapp"
  region = "us-east-1"
  count  = "5"
}

module "web-west" {
  source = "./webapp"
  region = "us-west-1a"
  count  = "10"
}
```

Modules

Outputs allow logically linking modules

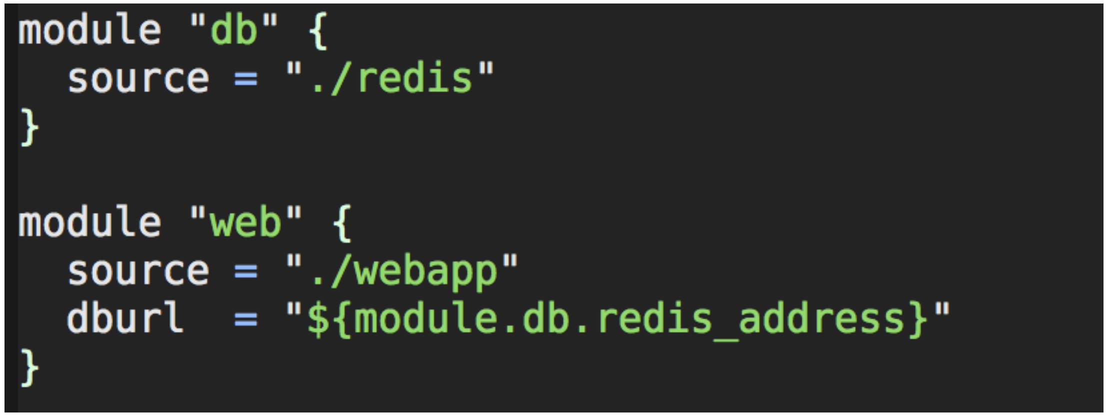


```
output "redis_address" {
  default = "${aws_instance.redis.public_ip}"
}
```

Can be thought of as the “return” value of a function.

Modules

Output values can be used as inputs to other resources or modules



```
module "db" {
  source = "./redis"
}

module "web" {
  source = "./webapp"
  dburl  = "${module.db.redis_address}"
}
```

Reliability

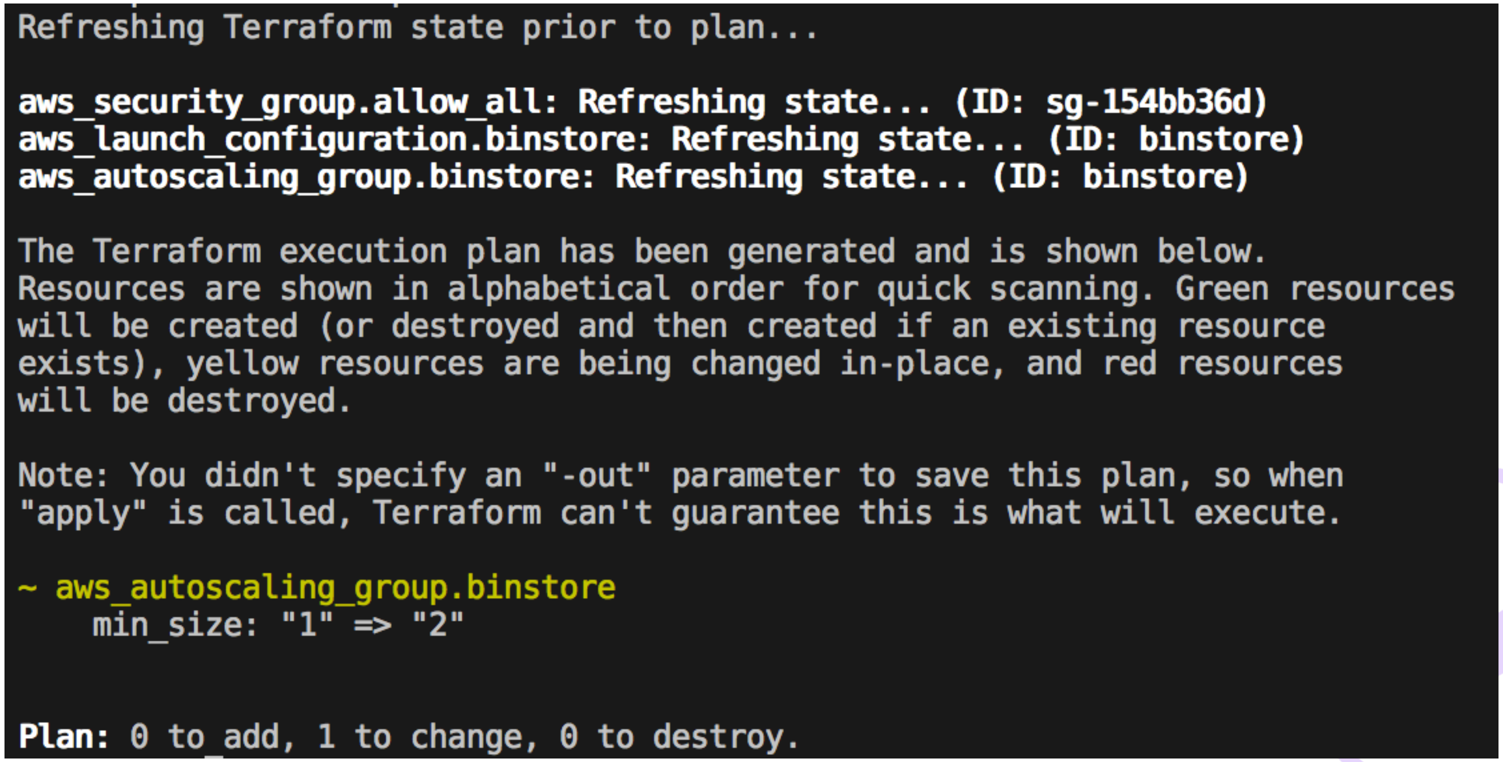
What happens if the ”world view” changes?

Reliability

What happens if the ”world view” changes?

Between separate “plan” runs:

- Terraform will refresh its state



```
Refreshing Terraform state prior to plan...

aws_security_group.allow_all: Refreshing state... (ID: sg-154bb36d)
aws_launch_configuration.binstore: Refreshing state... (ID: binstore)
aws_autoscaling_group.binstore: Refreshing state... (ID: binstore)

The Terraform execution plan has been generated and is shown below.
Resources are shown in alphabetical order for quick scanning. Green resources
will be created (or destroyed and then created if an existing resource
exists), yellow resources are being changed in-place, and red resources
will be destroyed.

Note: You didn't specify an "-out" parameter to save this plan, so when
"apply" is called, Terraform can't guarantee this is what will execute.

~ aws_autoscaling_group.binstore
    min_size: "1" => "2"


Plan: 0 to_add, 1 to change, 0 to destroy.
```

Reliability

What happens if the ”world view” changes?

Between a “plan” and an “apply”:

- Refreshes state, assumes it reflects the expected changes

- Better predictability by saving plans

```
> terraform plan –out terraform.tfplan
> terraform apply terraform.tfplan
```

Reliability

What happens if the apply fails?

- Terraform persists its state and exits
  - No automatic roll-back

- Lean on idempotency for recovery

Reliability

What happens if the state is lost?

Bad things . . .

Terraform can not “import” existing resources from infrastructure API’s (although this may come in the future).

Preventitive measures:

- Git or other VCS for local state
- Remote State (s3, Atlas, …)

Reliability

What happens if Terraform is interrupted?

- Partial state is still written
  - Each resource change recorded individually

- Terraform can continue from the last save

How HashiCorp uses Terraform

Logical component separation

- Modules used heavily to separate infrastructure concerns
  - Network
  - Storage
  - Compute

Decoupled from Credentials

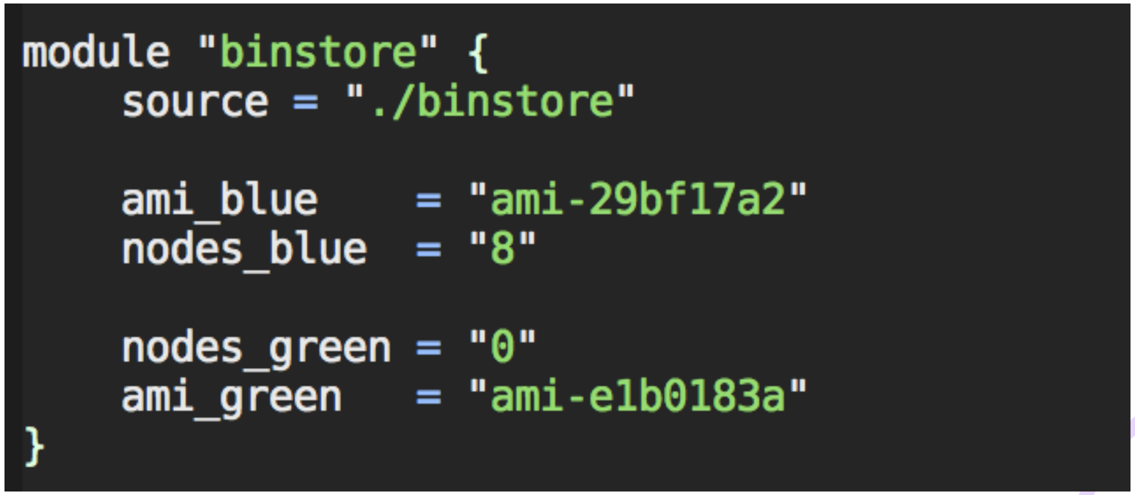
- Environment variables used to separate infrastructure code from sensitive credentials
- Makes duplicating environments to different accounts or regions easy

Remote State Only

- Remote state provides decentralized management abilities
- Durability and ease-of-access for critical state information
- Caveat: Time-of-check/time-of-use problem still exists

Blue/Green Deploys

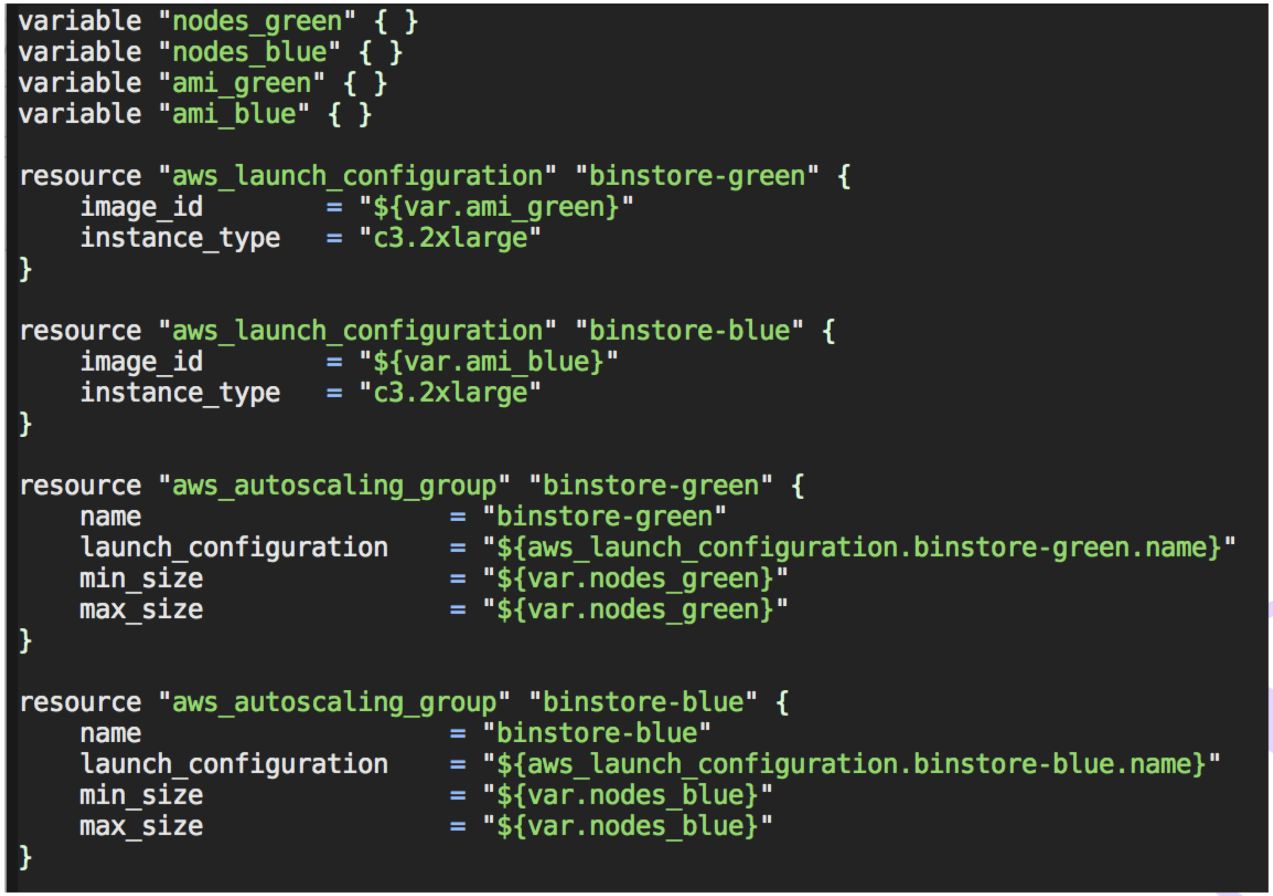
- Specify blue/green artifacts and counts as module parameters



```
module "binstore" {
    source = "./binstore"

    ami_blue    = "ami-29bf17a2"
    nodes_blue  = "8"

    nodes_green = "0"
    ami_green   = "ami-e1b0183a"
}
```

Blue/Green Deploys

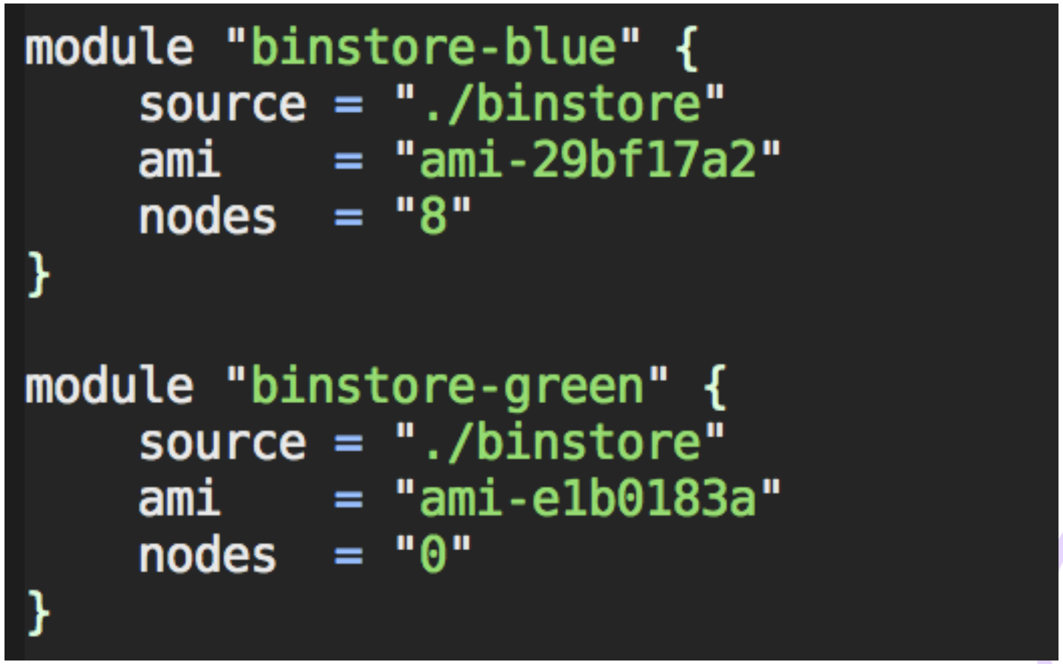
- Separate resource pools maintained for each group (blue/green)



```
variable "nodes_green" { }
variable "nodes_blue" { }
variable "ami_green" { }
variable "ami_blue" { }

resource "aws_launch_configuration" "binstore-green" {
    image_id        = "${var.ami_green}"
    instance_type   = "c3.2xlarge"
}

resource "aws_launch_configuration" "binstore-blue" {
    image_id        = "${var.ami_blue}"
    instance_type   = "c3.2xlarge"
}

resource "aws_autoscaling_group" "binstore-green" {
    name                    = "binstore-green"
    launch_configuration    = "${aws_launch_configuration.binstore-green.name}"
    min_size                = "${var.nodes_green}"
    max_size                = "${var.nodes_green}"
}

resource "aws_autoscaling_group" "binstore-blue" {
    name                    = "binstore-blue"
    launch_configuration    = "${aws_launch_configuration.binstore-blue.name}"
    min_size                = "${var.nodes_blue}"
    max_size                = "${var.nodes_blue}"
}
```

Blue/Green Deploys

Could also be written as separate module calls:



```
module "binstore-blue" {
    source = "./binstore"
    ami    = "ami-29bf17a2"
    nodes  = "8"
}

module "binstore-green" {
    source = "./binstore"
    ami    = "ami-e1b0183a"
    nodes  = "0"
}
```

Terraform Questions?

Consul

https://consul.io

What is Consul?

- Service Discovery
- Configuration Management
- Distributed, highly available, fault tolerant

- Service Discovery
- Configuration Management
- Distributed, highly available, fault tolerant

How do I connect things together?

Applications need configuration

Configuration is unknown prior to runtime

Configuration may change

Commonly Required Configuration

Hostname or IP address

Port number

Arbitrary, domain-specific metadata

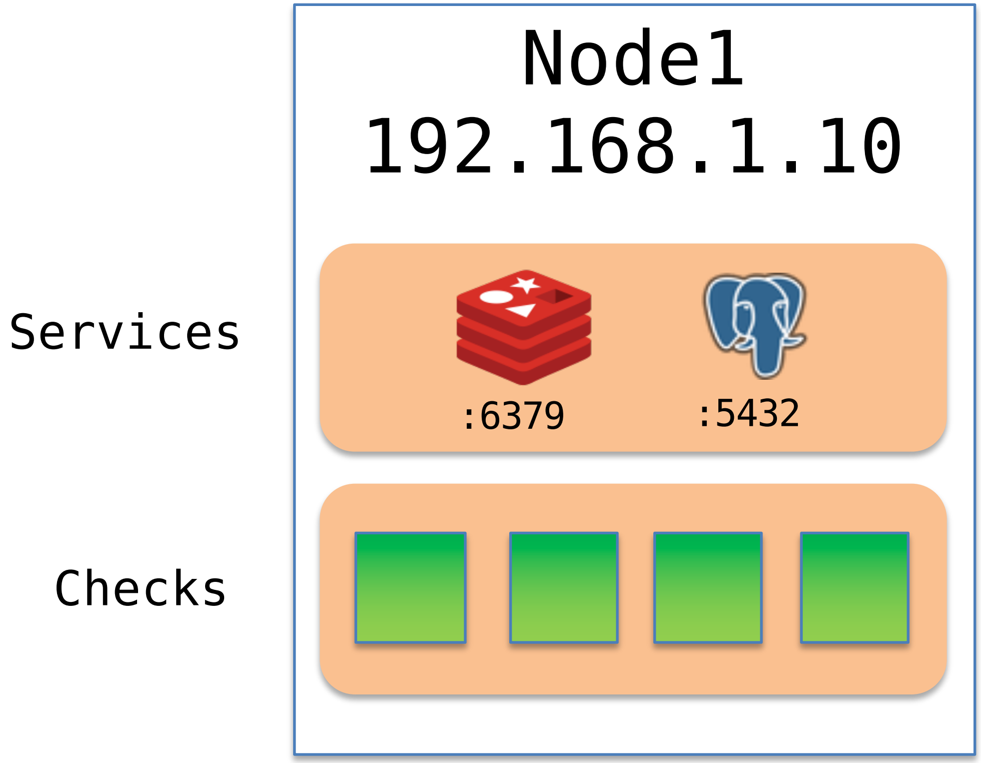
Core Consul Concepts

Nodes – Have IP addresses, services, and health status (CPU, Mem, etc.)

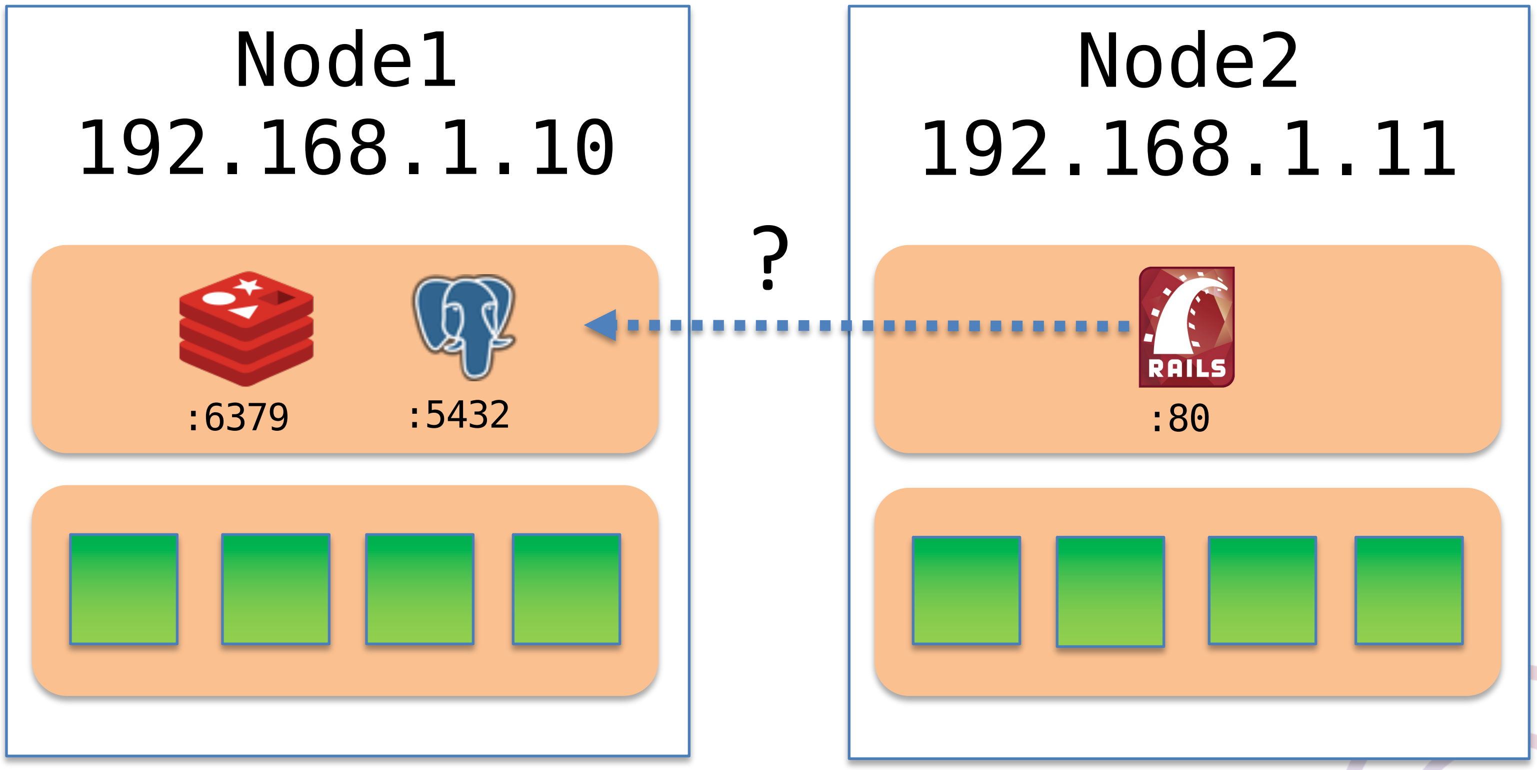
Services – Have logical names, port numbers, tags, and health status

Key/Value Pairs – Flat string-to-bytes mapping for arbitrary storage

Service Discovery



Service Discovery



Application Config

```
{
  “postgres_addr”: “???”,
  “redis_addr”: “???”
}
```

IP Address

- or -

DNS Hostname

Application Config

```
{
  “postgres_addr”: “pg.service.consul:5432”,
  “redis_addr”: “redis.service.consul:6379”
}
```

IP Address

- or -

DNS Hostname

Consul DNS

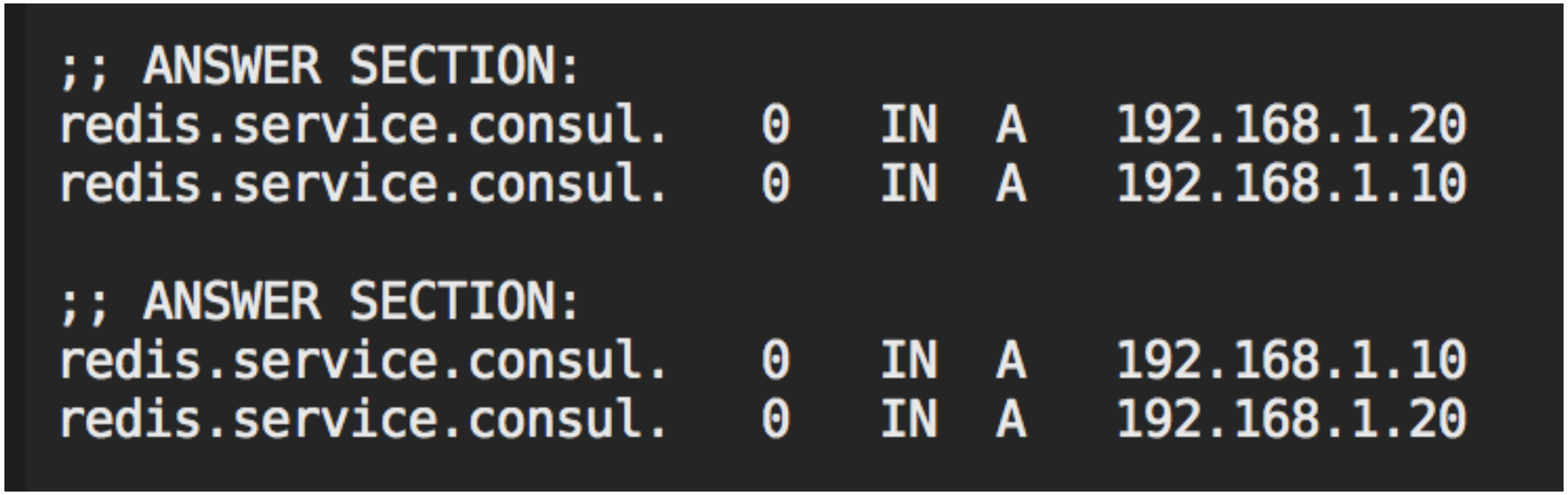
Expose nodes and services:

```
<nodeID>.node.consul
<serviceID>.service.consul
```

```
> dig +short redis.service.consul
192.168.1.10

> dig +short SRV redis.service.consul
1 1 6379 node1.node.dc1.consul.
```

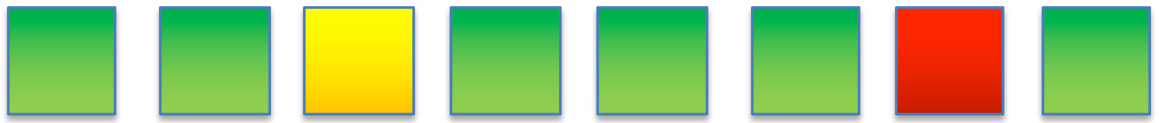
Consul DNS

Round Robin by default



```
;; ANSWER SECTION:
redis.service.consul.   0    IN   A    192.168.1.20
redis.service.consul.   0    IN   A    192.168.1.10

;; ANSWER SECTION:
redis.service.consul.   0    IN   A    192.168.1.10
redis.service.consul.   0    IN   A    192.168.1.20
```

Health Checks



Health Checks

Operational visibility to the emergent state of the cluster

Intelligently pair requests to services

Graceful degradation, maintenance windows

Health Checks

Check Types

- Basic script + interval (Nagios-compatible)
- HTTP/TCP
- TTL-based (dead man’s switch)

Health Checks

- Check workload handled collectively by the cluster
- Built-in Serf failure detector
- Check status affects service availability

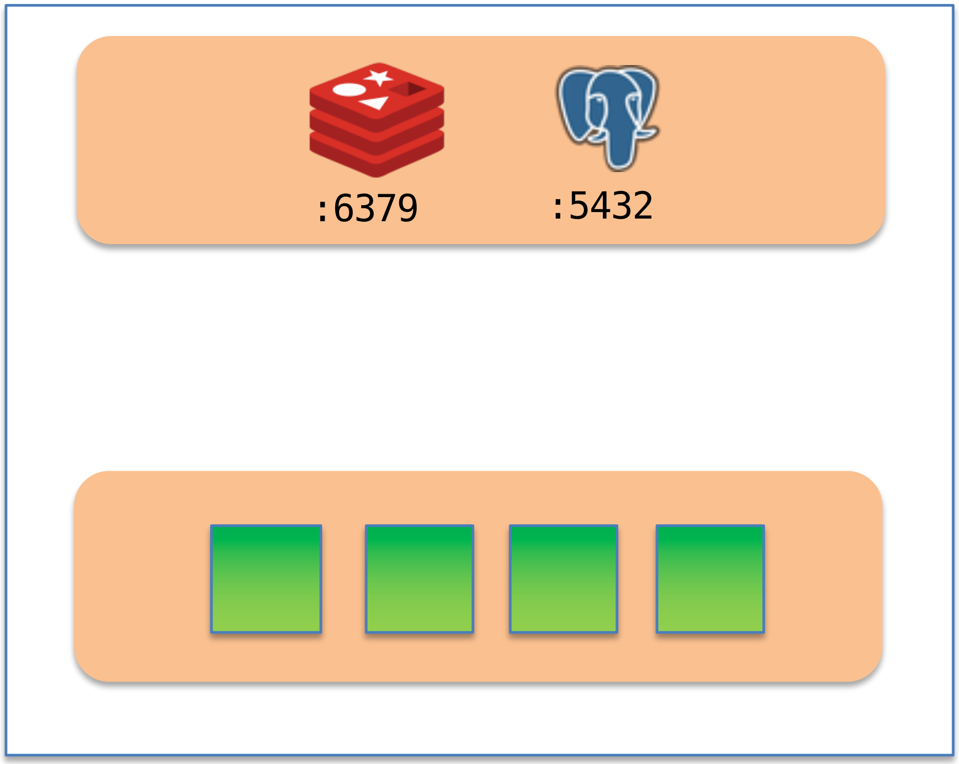
Health Checks

Check Scopes

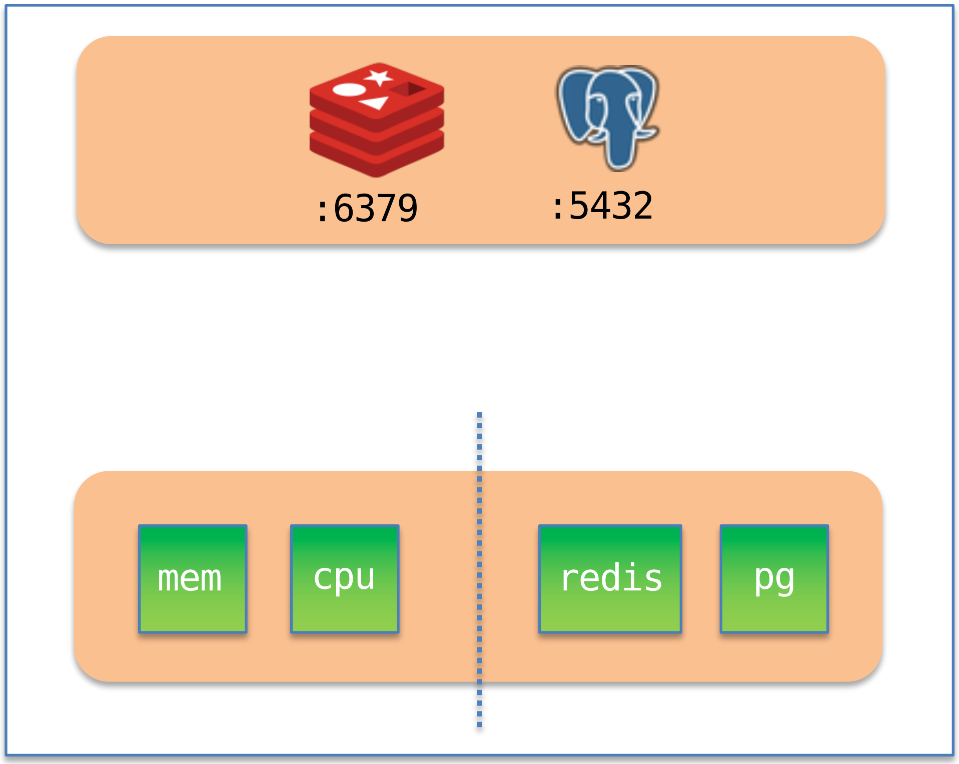
- Node – Affect availability of all services hosted on the node.
  Ex: “mem”, “disk”, “cpu”

- Service – Affect availability of only a specific service.
  Ex: “redis-tcp”

Health Checks

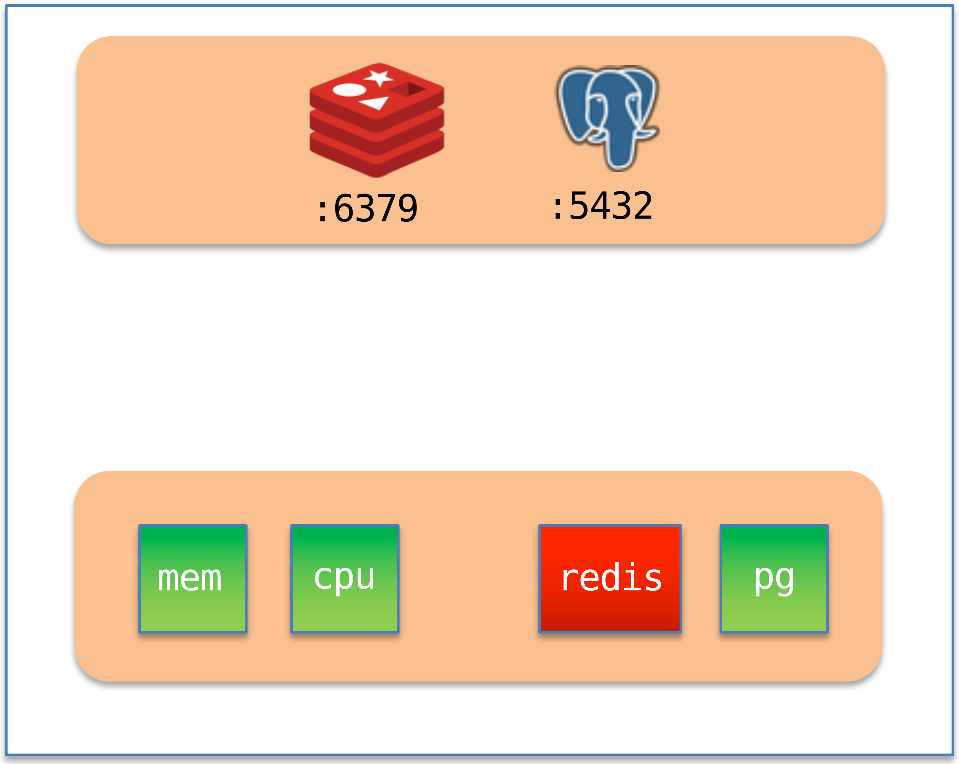


Health Checks

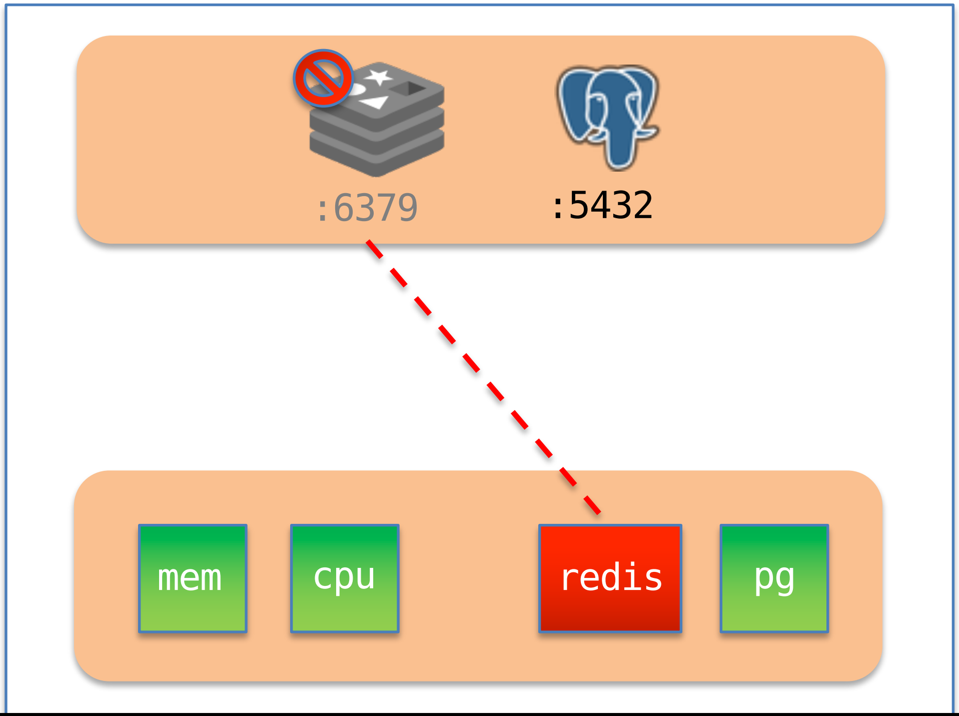


Node Checks   Service Checks

Health Checks

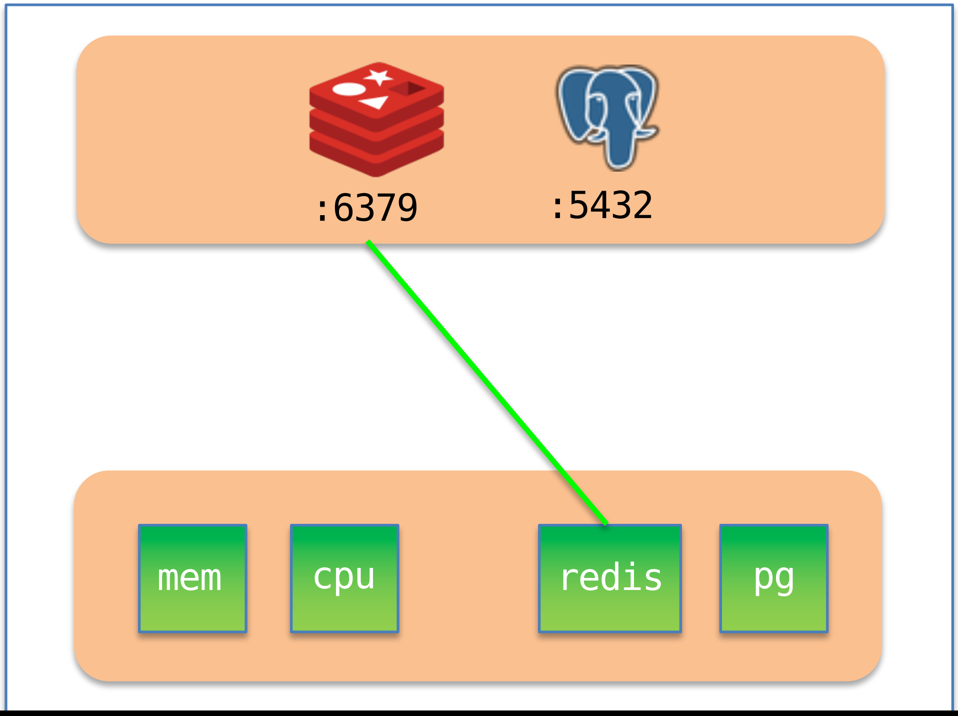


Health Checks



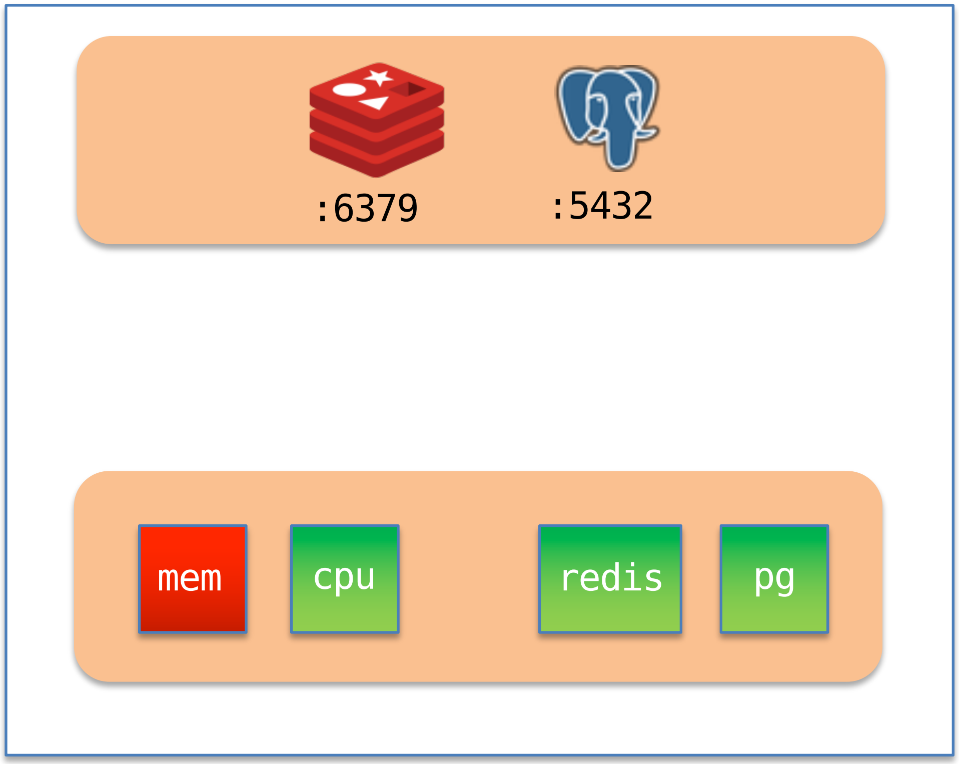
```
> dig redis.service.consul
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN
```

Health Checks

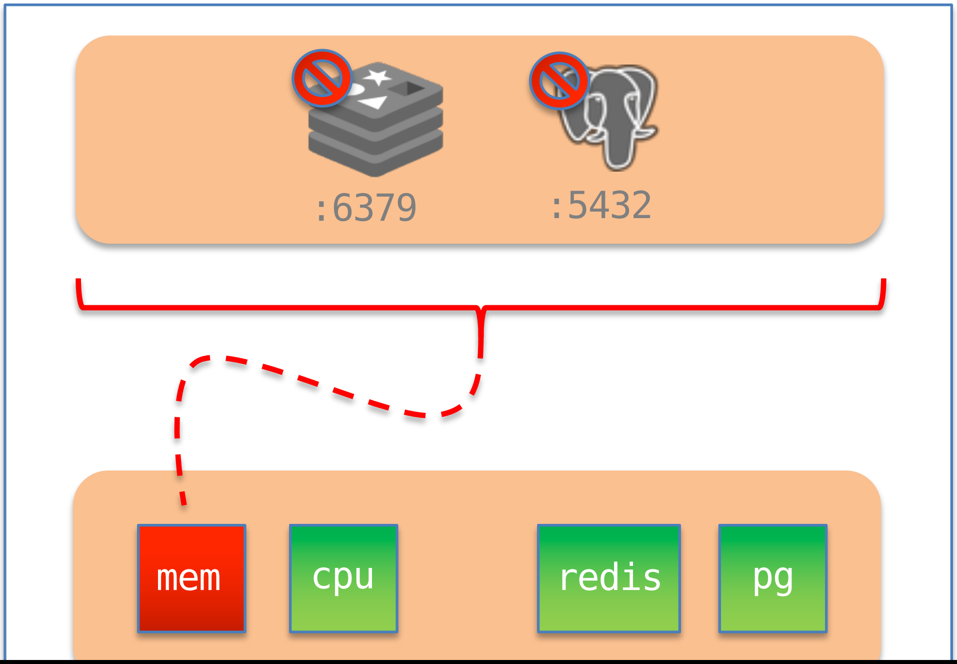


```
> dig +short redis.service.consul
192.168.1.10
```

Health Checks

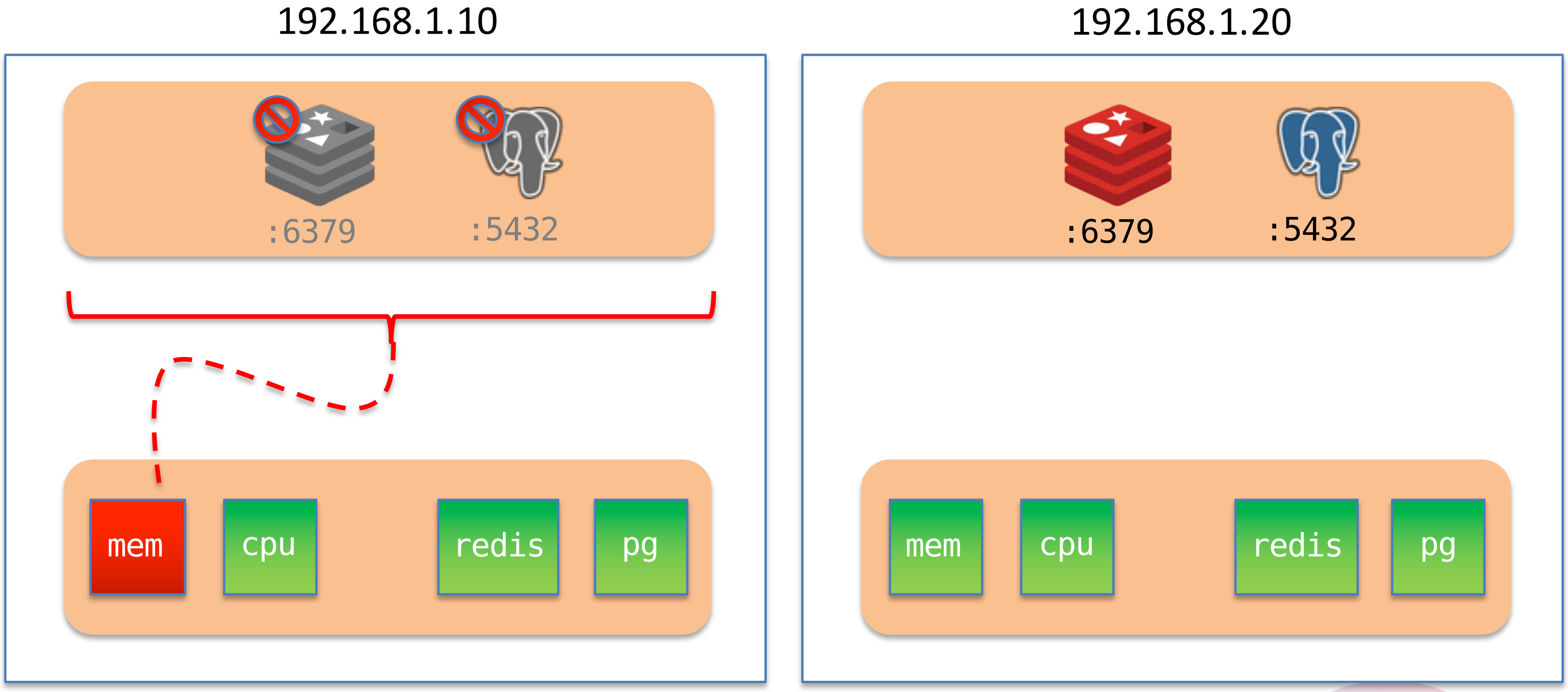


Health Checks



```
> dig redis.service.consul
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN
> dig pg.service.consul
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN
```

Health Checks



```
> dig +short redis.service.consul
192.168.1.20
```

- Service Discovery
- Configuration Management
- Distributed, highly available, fault tolerant

Config Management at Runtime

- Applications may have domain-specific configuration.
- Immutable configuration costs time
- Manual operator intervention is error prone
- Inter-node orchestration may be required

Consul Key/Value Store

Simple input/output over HTTP

```
> curl -X PUT localhost:8500/v1/kv/foo -d bar
true

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

Consul Key/Value Store

Blocking queries (HTTP long-poll)

```
> curl -i localhost:8500/v1/kv/foo?raw
X-Consul-Index: 541
bar

> curl localhost:8500/v1/kv/foo?raw&index=541
... Time passes ...
baz
```

```
> curl -X PUT localhost:8500/v1/kv/foo -d baz
true
```

Consul Key/Value Store

Long-poll Limitations

- Change is not guaranteed
- Deduplication on client side
- Full-scope response payload
- Data safety handled by client

Sessions and Locks

Provide mutual exclusion and semaphore primitives

```
                                   Create session
> curl -X PUT localhost:8500/v1/session/create
{"ID":"179c685c-179d-a186-ba71-920952e8428c"}

                                     Acquire lock
> curl -X PUT localhost:8500/v1/kv/foo
?acquire= 179c685c-179d-a186-ba71-920952e8428c

                                     Release lock
> curl -X PUT localhost:8500/v1/kv/foo
?release= 179c685c-179d-a186-ba71-920952e8428c
```

Session Invalidation

- Sessions may be linked to checks
- Sessions may provide a TTL

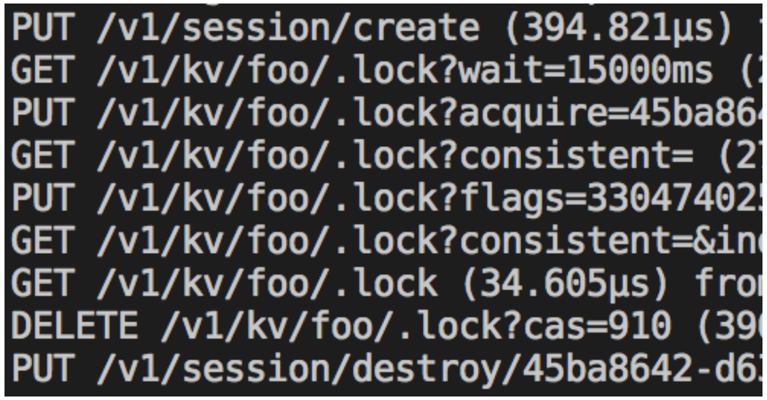
```
> curl -X PUT localhost:8500/v1/session/create \
-d ‘{
  "LockDelay": "15s",
  "Node": ”Node1",
  "Checks": [”serfHealth", ”mem", "cpu"],
  "Behavior": "release",
  "TTL": ”1h”
}’
```

Prevents unhealthy nodes from holding a lock

“consul lock”

Wraps session creation, key locking and releasing around a process.

```
> consul lock foo “echo hello”
hello
```



```
PUT /v1/session/create (394.821µs)
GET /v1/kv/foo/.lock?wait=15000ms (
PUT /v1/kv/foo/.lock?acquire=45ba86
GET /v1/kv/foo/.lock?consistent= (2
PUT /v1/kv/foo/.lock?flags=33047402
GET /v1/kv/foo/.lock?consistent=&in
GET /v1/kv/foo/.lock (34.605µs) fro
DELETE /v1/kv/foo/.lock?cas=910 (39
PUT /v1/session/destroy/45ba8642-d6
```

Supports multiple holders with “-n” flag

“consul lock”

Useful for rolling deploys/restarts

Fully serialized restarts

```
> consul lock foo “restart binstore”
binstore start/running, process 3004
```

Multiple parallel restarts

```
> consul lock foo -n 2 “restart binstore”
binstore start/running, process 3004
```

envconsul

https://github.com/hashicorp/envconsul

Bridge Consul K/V and 12-factor apps

Export K/V pairs as environment vars

```
> envconsul \
    -prefix ”service/binstore” \
    /usr/local/bin/binstore

2016/02/17 12:16:17 [DEBUG] Starting server...
```

consul-template

https://github.com/hashicorp/consul-template

Render config file templates from Consul data

```
{
  “posgres_addr”: “{{key “service/pg/addr”}}”,
  “redis_addr”: “{{key “service/redis/addr”}}”
}
```

consul-template

https://github.com/hashicorp/consul-template

First-class services integration

```
listen web-proxy 0.0.0.0:80
  mode http
  balance roundrobin
{{range service “binstore”}}
  server {{.Node}} {{.Address}}:{{.Port}}
{{end}}
```

- Service Discovery
- Configuration Management
- Distributed, highly available, fault tolerant

Serf for Cluster Membership

https://serfdom.io

- Gossip-based (SWIM) for scalable cluster convergence
- Fast failure detection
- Efficient event distribution

Raft for consensus and replication

Strongly consistent writes

Log replication

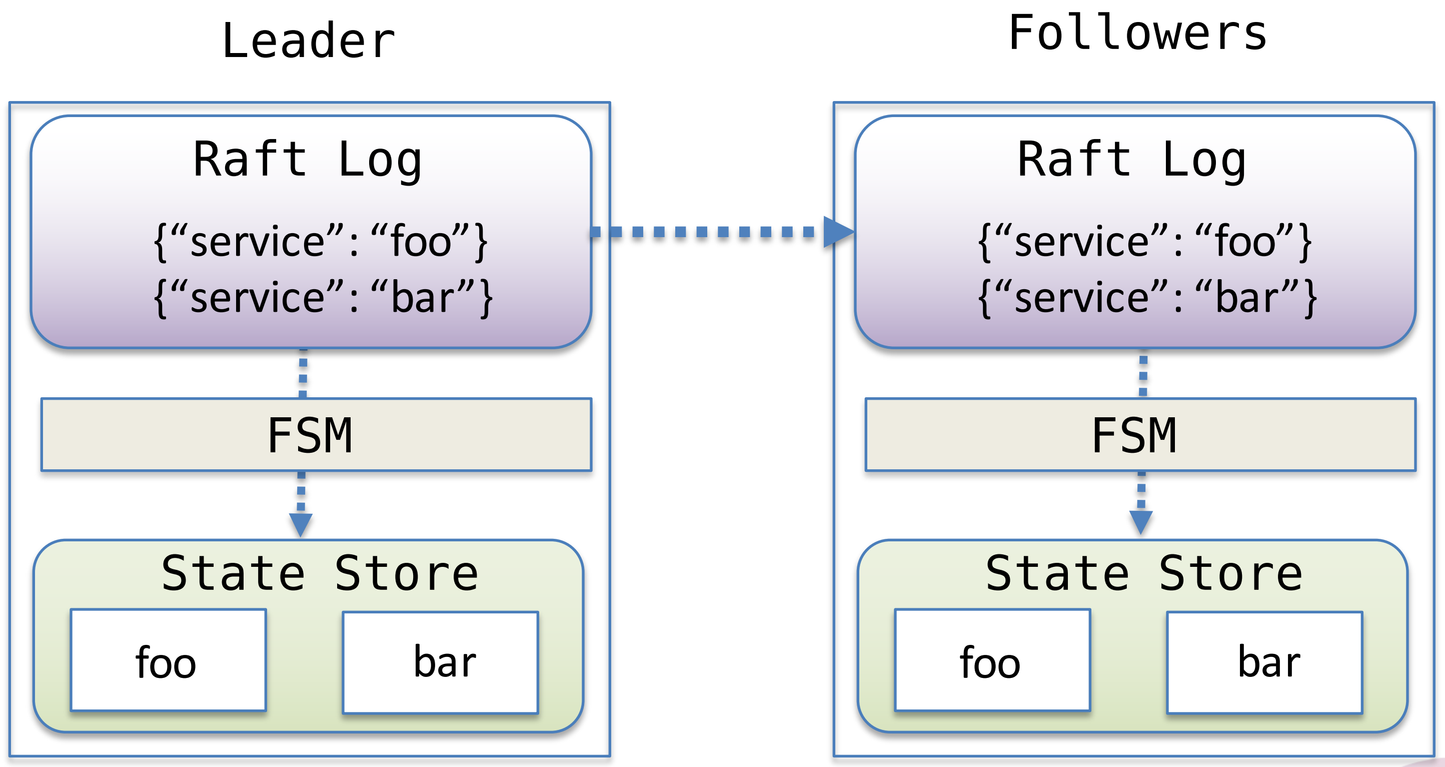
Fault tolerance

Raft Trade-offs

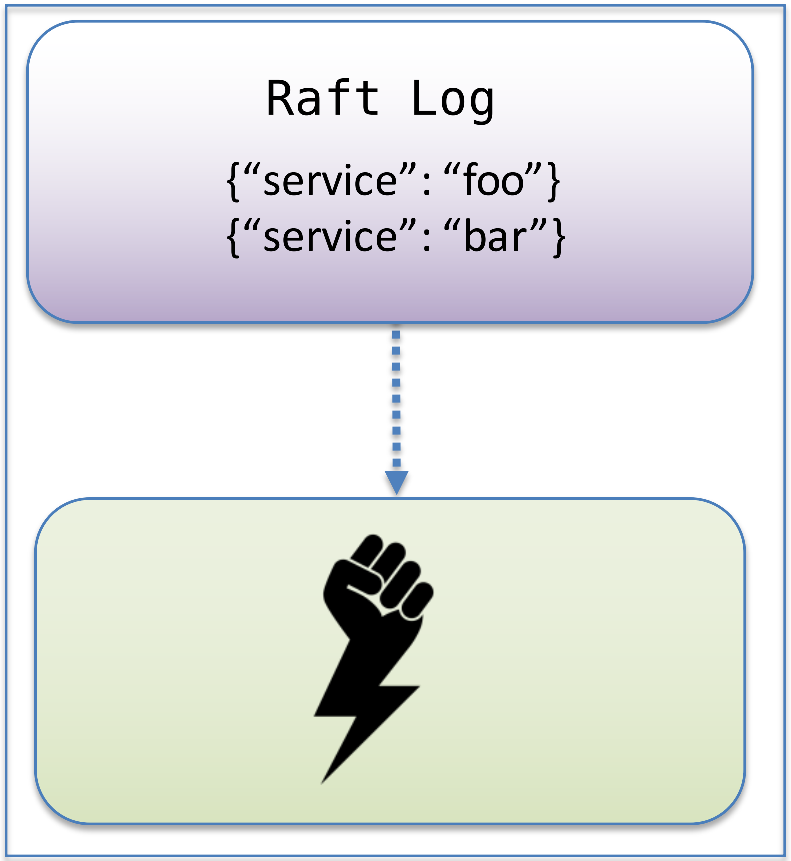
```
More peers = Higher fault tolerance

More peers = Higher consensus complexity,
             slower write performance.
```

# of Peers	Fault Tolerance	Quorum Size
3	1	2
5	2	3
7	3	4

Replication in Consul



BoltDB for durable storage



Log recovery in outage scenarios

Fast, pure-Go on-disk K/V storage

MemDB for state storage

In-memory ephemeral store

Indexes native types for speed

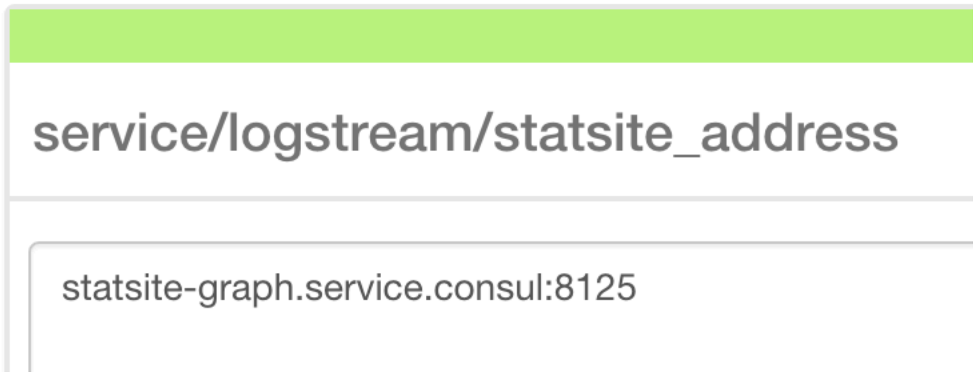
Provides fast stale-read access from any server node

How HashiCorp uses Consul

K/V store for configuration

Store **all** configuration values in K/V

Even Consul-generated DNS names



service/logstream/statsite_address

statsite-graph.service.consul:8125

Agent on every machine

- Consul runs in client-only mode on all nodes.
- Distributes workload, Makes querying easy.
- Exposes node outages
- Enables practical use of “consul lock”

Various Niceties

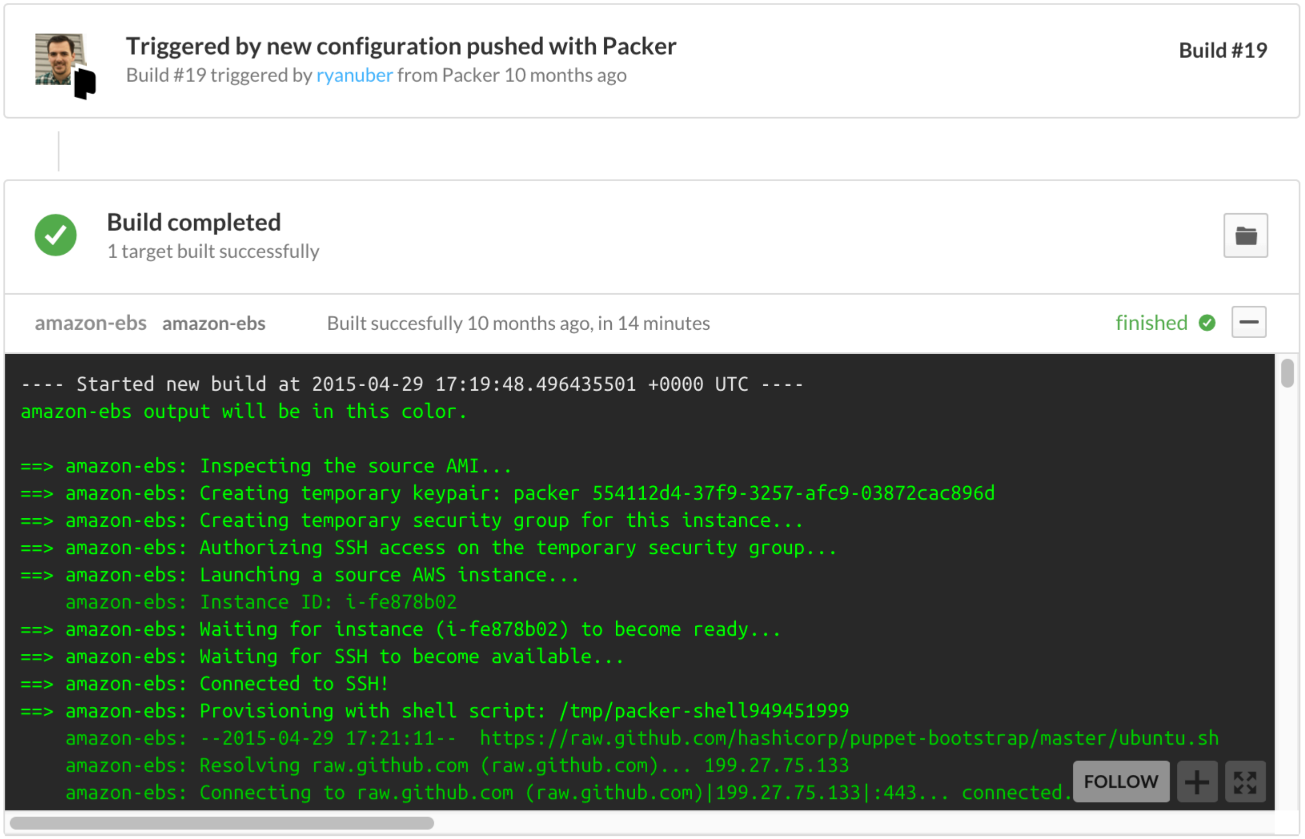
- Use /etc/consul.d/ to drop in service configs
- Configure VPN to use Consul DNS
- DSH + consul instead of “consul exec”

Consul Questions?

Atlas

https://atlas.hashicorp.com

Packer build monitoring



Triggered by new configuration pushed with Packer
Build #19 triggered by ryanuber from Packer 10 months ago

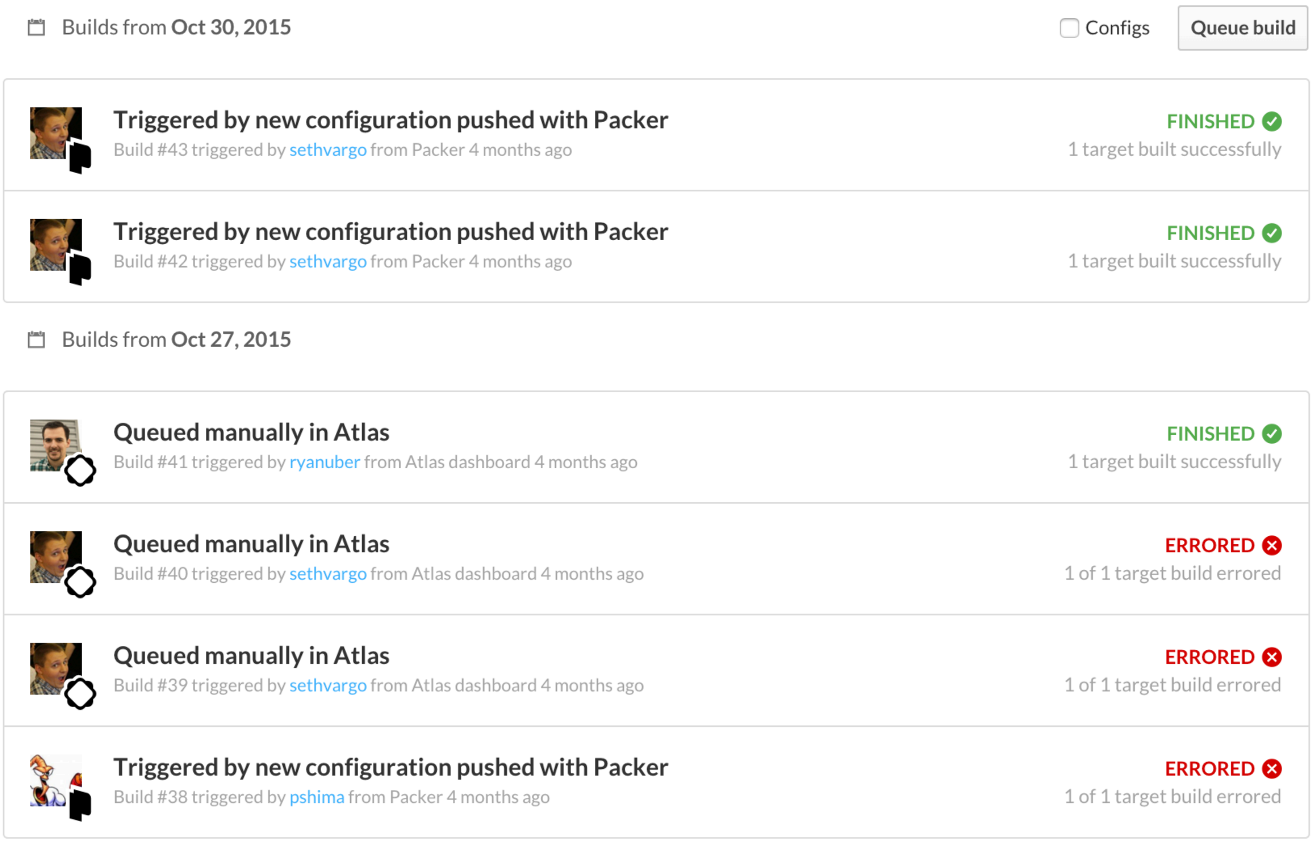
Build #19

Build completed
1 target built successfully

amazon-ebs amazon-ebs Built succesfully 10 months ago, in 14 minutes finished ✓

```
---- Started new build at 2015-04-29 17:19:48.496435501 +0000 UTC ----
amazon-ebs output will be in this color.

==> amazon-ebs: Inspecting the source AMI...
==> amazon-ebs: Creating temporary keypair: packer 554112d4-37f9-3257-afc9-03872cac896d
==> amazon-ebs: Creating temporary security group for this instance...
==> amazon-ebs: Authorizing SSH access on the temporary security group...
==> amazon-ebs: Launching a source AWS instance...
    amazon-ebs: Instance ID: i-fe878b02
==> amazon-ebs: Waiting for instance (i-fe878b02) to become ready...
==> amazon-ebs: Waiting for SSH to become available...
==> amazon-ebs: Connected to SSH!
==> amazon-ebs: Provisioning with shell script: /tmp/packer-shell949451999
    amazon-ebs: --2015-04-29 17:21:11--  https://raw.github.com/hashicorp/puppet-bootstrap/master/ubuntu.sh
    amazon-ebs: Resolving raw.github.com (raw.github.com)... 199.27.75.133
    amazon-ebs: Connecting to raw.github.com (raw.github.com)|199.27.75.133|:443... connected.
```

Packer build history



Builds from **Oct 30, 2015**

Triggered by new configuration pushed with Packer
Build #43 triggered by sethvargo from Packer 4 months ago
FINISHED ✓
1 target built successfully

Triggered by new configuration pushed with Packer
Build #42 triggered by sethvargo from Packer 4 months ago
FINISHED ✓
1 target built successfully

Builds from **Oct 27, 2015**

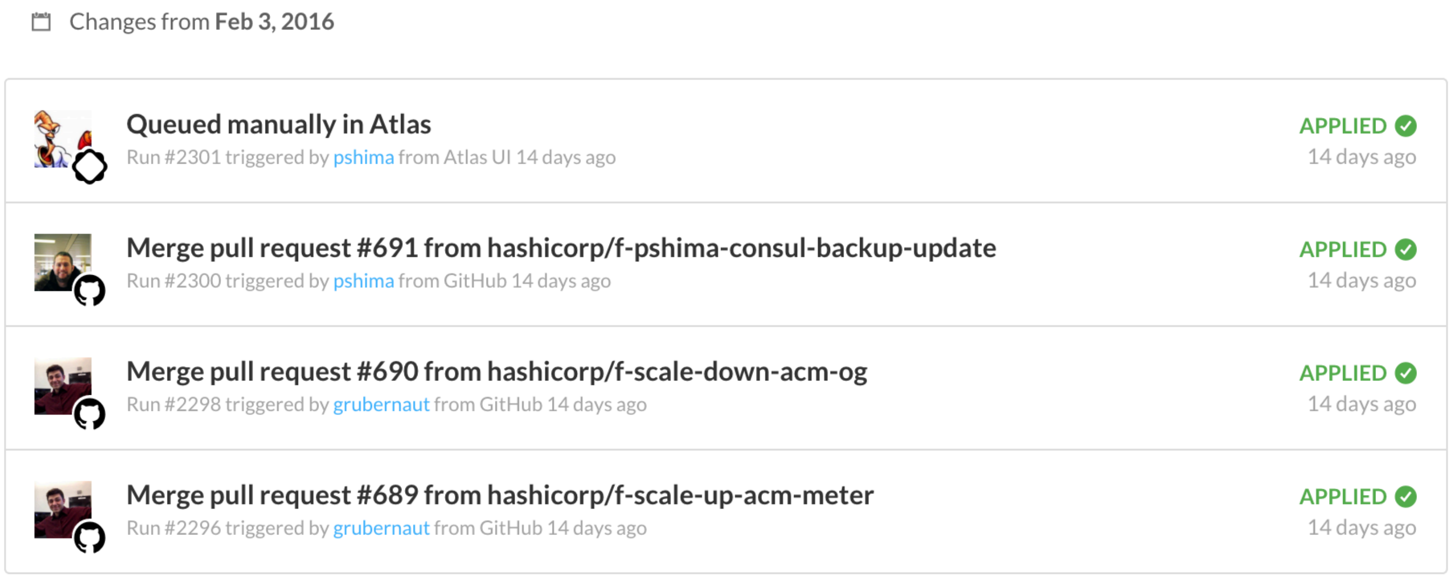
Queued manually in Atlas
Build #41 triggered by ryanuber from Atlas dashboard 4 months ago
FINISHED ✓
1 target built successfully

Queued manually in Atlas
Build #40 triggered by sethvargo from Atlas dashboard 4 months ago
ERRORED ✗
1 of 1 target build errored

Queued manually in Atlas
Build #39 triggered by sethvargo from Atlas dashboard 4 months ago
ERRORED ✗
1 of 1 target build errored

Triggered by new configuration pushed with Packer
Build #38 triggered by pshima from Packer 4 months ago
ERRORED ✗
1 of 1 target build errored

Terraform Change History



Changes from **Feb 3, 2016**

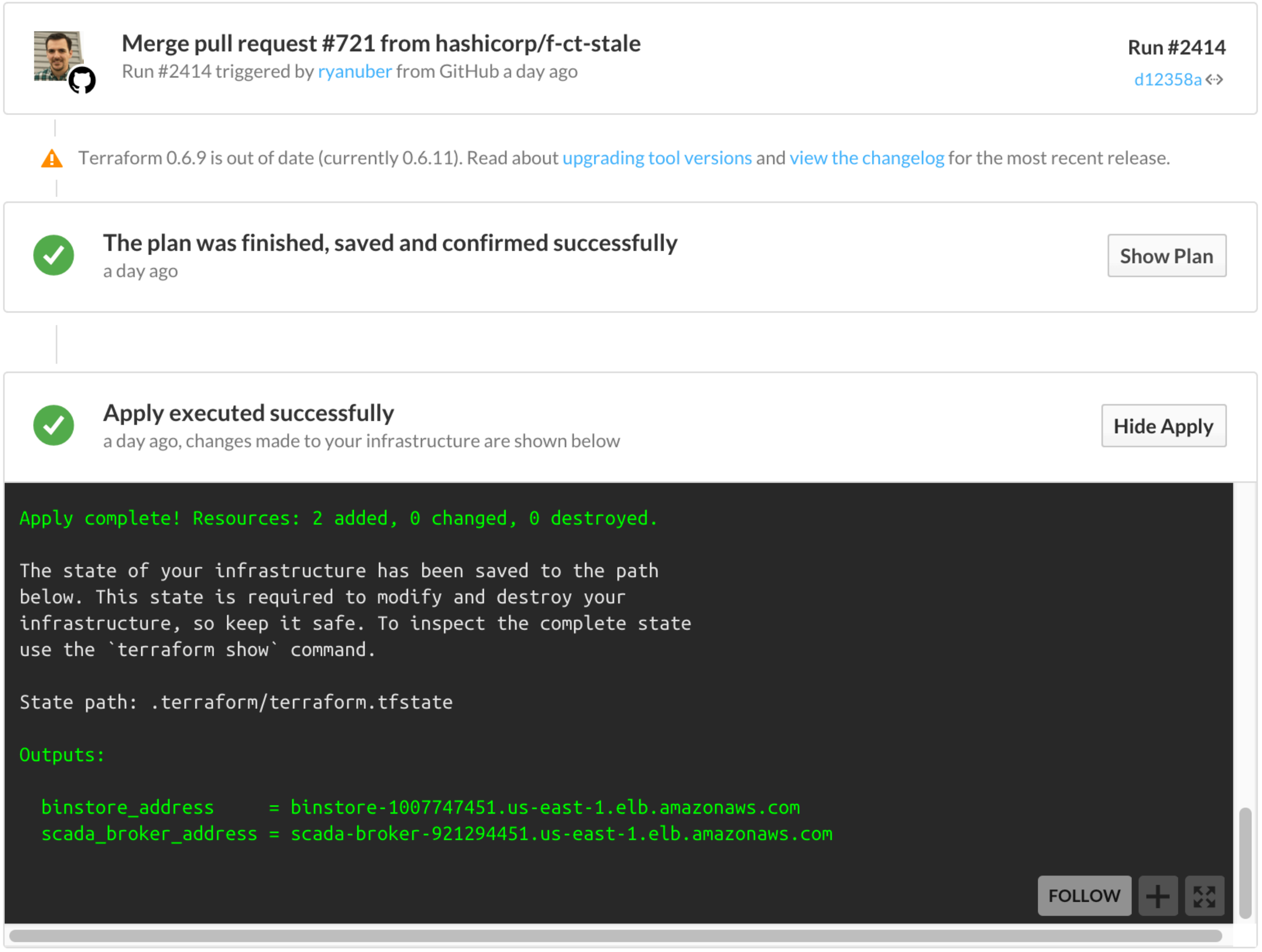
Queued manually in Atlas
Run #2301 triggered by pshima from Atlas UI 14 days ago
APPLIED ✓
14 days ago

Merge pull request #691 from hashicorp/f-pshima-consul-backup-update
Run #2300 triggered by pshima from GitHub 14 days ago
APPLIED ✓
14 days ago

Merge pull request #690 from hashicorp/f-scale-down-acm-og
Run #2298 triggered by grubernaut from GitHub 14 days ago
APPLIED ✓
14 days ago

Merge pull request #689 from hashicorp/f-scale-up-acm-meter
Run #2296 triggered by grubernaut from GitHub 14 days ago
APPLIED ✓
14 days ago

Terraform Run Monitoring



Merge pull request #721 from hashicorp/f-ct-stale
Run #2414 triggered by ryanuber from GitHub a day ago

Run #2414
d12358a

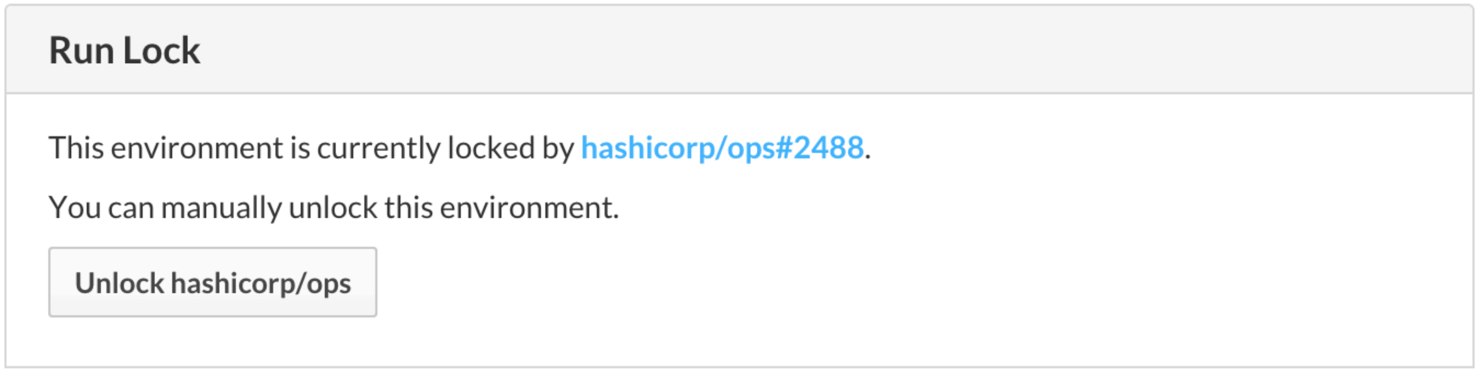
Terraform 0.6.9 is out of date (currently 0.6.11). Read about upgrading tool versions and view the changelog for the most recent release.

The plan was finished, saved and confirmed successfully
a day ago

Apply executed successfully
a day ago, changes made to your infrastructure are shown below

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
infrastructure, so keep it safe. To inspect the complete state
use the `terraform show` command.

State path: .terraform/terraform.tfstate

Outputs:

  binstore_address     = binstore-1007747451.us-east-1.elb.amazonaws.com
  scada_broker_address = scada-broker-921294451.us-east-1.elb.amazonaws.com
```

Terraform Run Lock

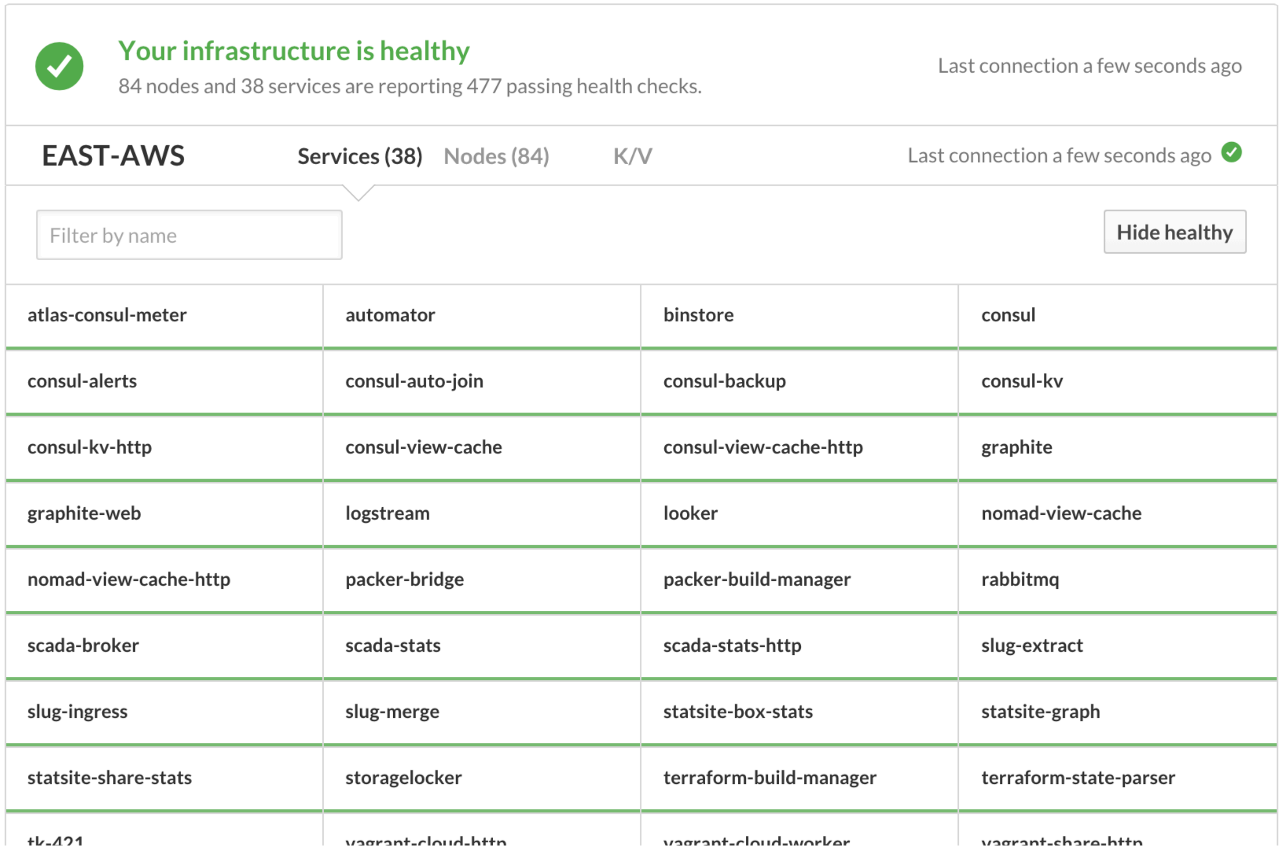


Run Lock

This environment is currently locked by hashicorp/ops#2488.

You can manually unlock this environment.

Enhanced Consul UI


Your infrastructure is healthy

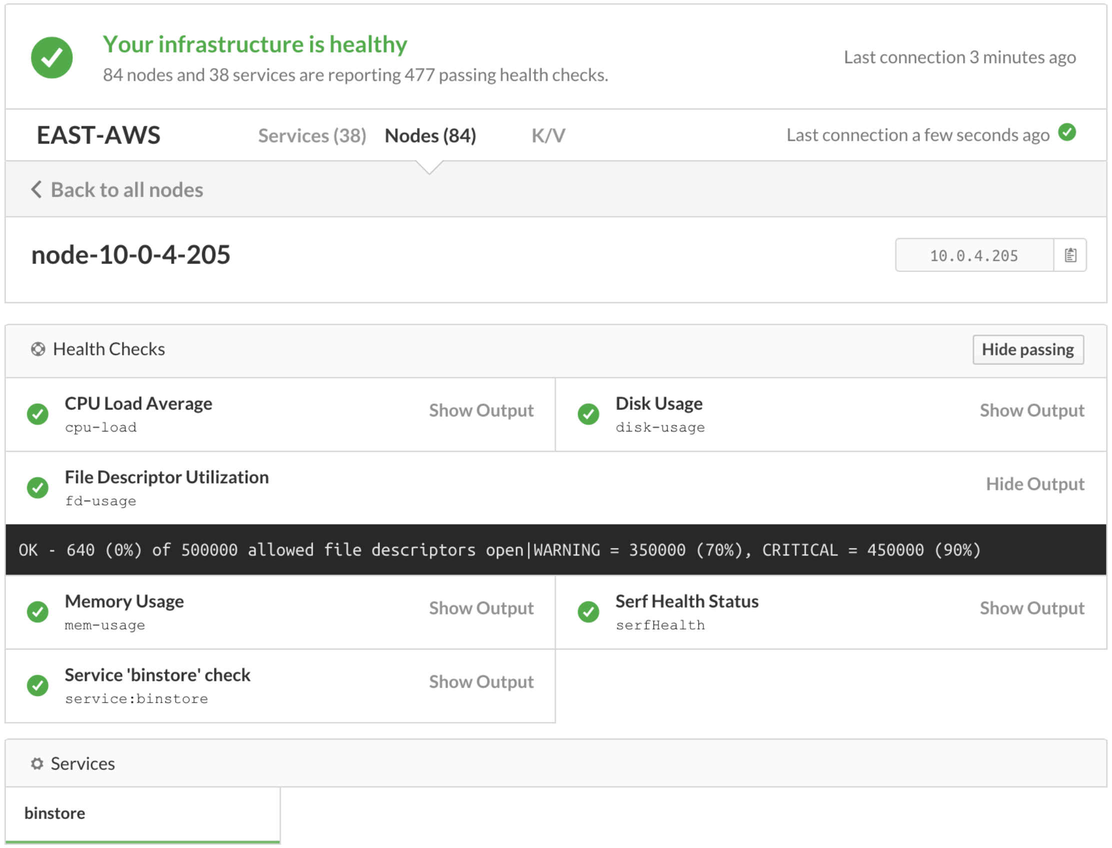
84 nodes and 38 services are reporting 477 passing health checks.

Last connection a few seconds ago

EAST-AWS — Services (38) · Nodes (84) · K/V — Last connection a few seconds ago ✓

atlas-consul-meter	automator	binstore	consul
consul-alerts	consul-auto-join	consul-backup	consul-kv
consul-kv-http	consul-view-cache	consul-view-cache-http	graphite
graphite-web	logstream	looker	nomad-view-cache
nomad-view-cache-http	packer-bridge	packer-build-manager	rabbitmq
scada-broker	scada-stats	scada-stats-http	slug-extract
slug-ingress	slug-merge	statsite-box-stats	statsite-graph
statsite-share-stats	storagelocker	terraform-build-manager	terraform-state-parser
tk-421	vagrant-cloud-http	vagrant-cloud-worker	vagrant-share-http

Enhanced Consul UI


Your infrastructure is healthy
84 nodes and 38 services are reporting 477 passing health checks.
Last connection 3 minutes ago
EAST-AWS
Services (38)
Nodes (84)
K/V
Last connection a few seconds ago
Back to all nodes

node-10-0-4-205

10.0.4.205

Health Checks

CPU Load Average
`cpu-load`

Disk Usage
`disk-usage`

File Descriptor Utilization
`fd-usage`

```
OK - 640 (0%) of 500000 allowed file descriptors open|WARNING = 350000 (70%), CRITICAL = 450000 (90%)
```

Memory Usage
`mem-usage`

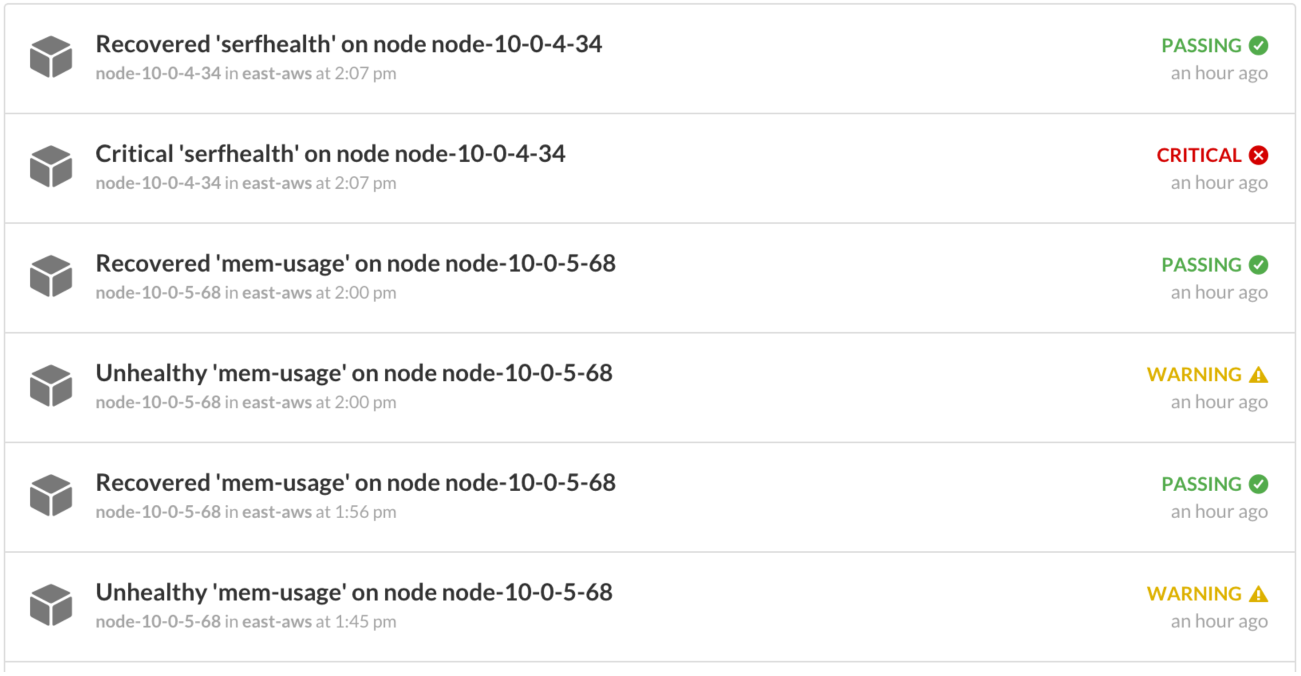
Serf Health Status
`serfHealth`

Service 'binstore' check
`service:binstore`

Services

binstore

Consul Alerts


Recovered 'serfhealth' on node node-10-0-4-34
node-10-0-4-34 in east-aws at 2:07 pm
PASSING ✓
an hour ago

Critical 'serfhealth' on node node-10-0-4-34
node-10-0-4-34 in east-aws at 2:07 pm
CRITICAL ✗
an hour ago

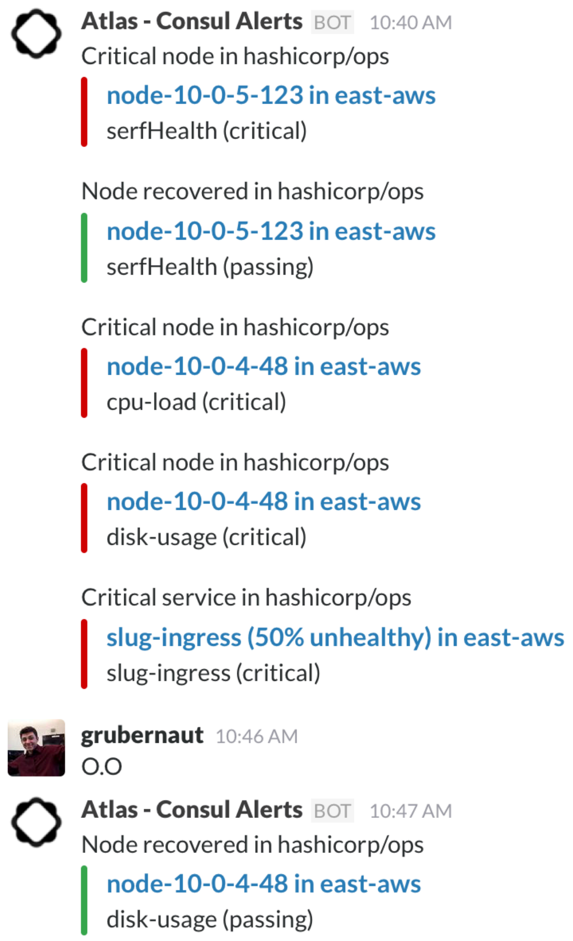
Recovered 'mem-usage' on node node-10-0-5-68
node-10-0-5-68 in east-aws at 2:00 pm
PASSING ✓
an hour ago

Unhealthy 'mem-usage' on node node-10-0-5-68
node-10-0-5-68 in east-aws at 2:00 pm
WARNING ⚠
an hour ago

Recovered 'mem-usage' on node node-10-0-5-68
node-10-0-5-68 in east-aws at 1:56 pm
PASSING ✓
an hour ago

Unhealthy 'mem-usage' on node node-10-0-5-68
node-10-0-5-68 in east-aws at 1:45 pm
WARNING ⚠
an hour ago

Consul Alerts Integrations



Atlas - Consul Alerts BOT 10:40 AM
Critical node in hashicorp/ops
node-10-0-5-123 in east-aws
serfHealth (critical)

Node recovered in hashicorp/ops
node-10-0-5-123 in east-aws
serfHealth (passing)

Critical node in hashicorp/ops
node-10-0-4-48 in east-aws
cpu-load (critical)

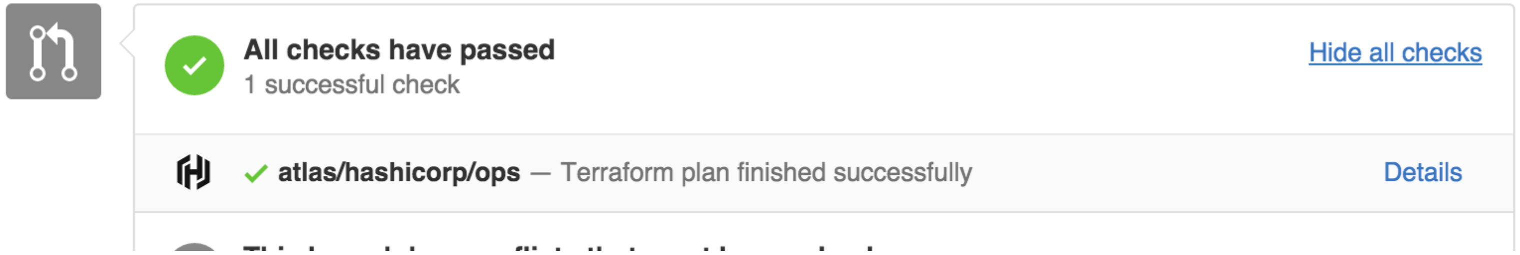
Critical node in hashicorp/ops
node-10-0-4-48 in east-aws
disk-usage (critical)

Critical service in hashicorp/ops
slug-ingress (50% unhealthy) in east-aws
slug-ingress (critical)

grubernaut 10:46 AM
O.O

Atlas - Consul Alerts BOT 10:47 AM
Node recovered in hashicorp/ops
node-10-0-4-48 in east-aws
disk-usage (passing)

GitHub Integration



All checks have passed
1 successful check

Hide all checks

✓ **atlas/hashicorp/ops** — Terraform plan finished successfully

Details

Questions?

Thank you!

Now come get some stickers!