The Chicken or the Egg

*Or.. how we created a **Stable** Preprod Environment as a Public Cloud Provider*

$(whoami)
Christina M. Pukropski

Senior SRE @ Akamai

- 8y at Linode, acquired by Akamai
- Philadelphia Native
- Held various roles and worked on almost every product
- US Lead of *In Reach* ERG
- Current Tech Lead of Virtualization and Host Platforms Team
- Engineer, runner, hiker, DIYer enthusiast, musician, birder, Auntie
- Cat Mom

Why am I here today?

Everyone has a story to tell, and sometimes it needs to be shared with the world...

Overview: The Problem

- Prior Test Datacenter Limitations
  - Reliability issues
  - Support “best effort” since there was no dedicated team
  - Not isolated completely from production
  - Only useful to a handful of Python application teams
  - SRE, Ops, Hardware had no safe place to test
- We needed a shared place to test real world changes
- Little precedent in the industry

The Journey

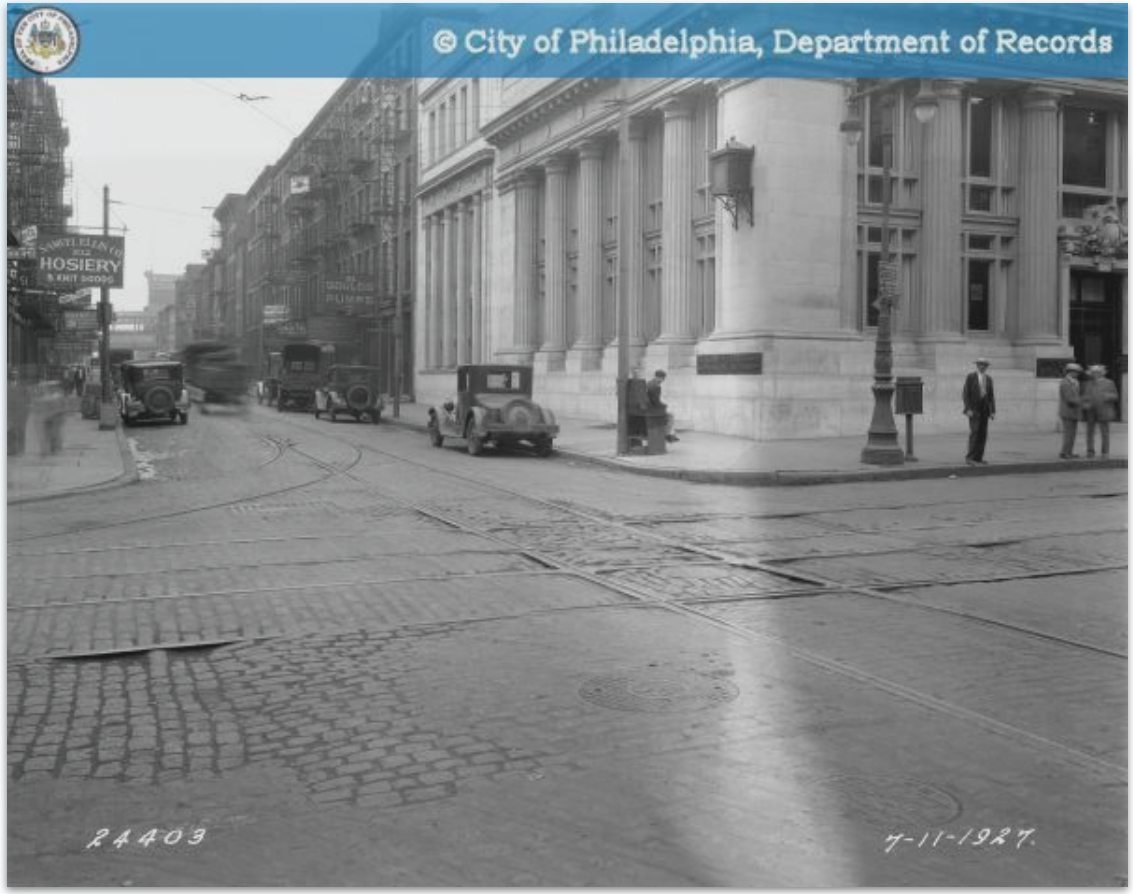
- **Overview**
- Building an Internal Testcloud
- First Customers
- Wins and Lessons Learned

First, Some History

- Cloud Company acquired by large public CDN company to be the branch into the compute business
- Had to scale very quickly!

Overview: History

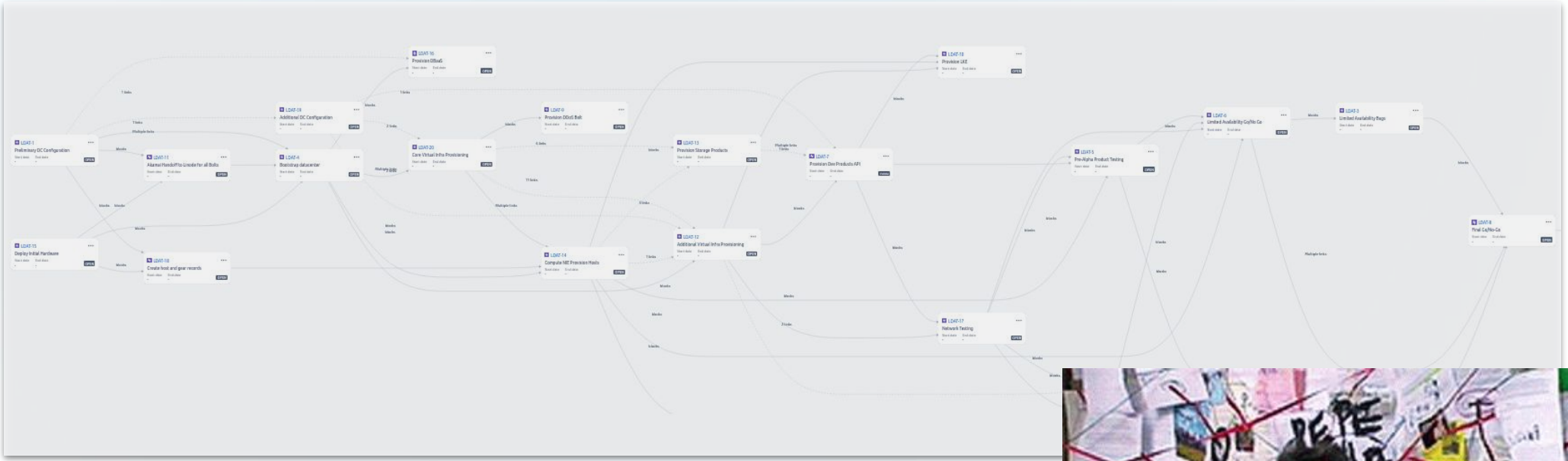
- 1.5y post-acquisition beginning of Testcloud project
- Year prior we deployed 13 datacenters
  - Doubled our compute footprint
- Many “chicken and egg” problems to solve
- Order of operations challenges



Overview: History

- Eliminated manual tasks and built into Terraform
- Various scaling Platform challenges
- Created process *Linode Datacenter Automation Template* “LDAT” and controlled through a standardized set of tasks that mapped linear dependencies
  - Allowed us to see
    - what could be run in parallel
    - what needed to be automated
  - Hacking Jira to use as a “relational database” for a pure “waterfall” Hardware project

Overview: Dependencies have Dependencies



Overview: Constraints

- Very small core team (6)
- Mix of SRE and Dev with various backgrounds and degrees of understanding of the stack
- Nothing inherited from production (total isolation)
- No sharing of resources from production

Overview: Requirements

- Replace and deprecate existing Test Datacenter
- Adapt to new version of our network stack
- Deployed in a private RFC1918 IP space within Akamai’s lab network

Overview: Requirements

- Simulate a realistic “load” of an actual Datacenter
- Test various parts of the stack in shared environment from Hardware to Software
- Stack to be “stable” as well as the main point for “real” integration
- **Need to maintain parity**

… so just make a cloud from scratch

The Journey

- Overview
- **Building an Internal Testcloud**
- First Customers
- Wins and Lessons Learned

Building an Internal Testcloud

- Definitions
  - Hypervisor - runs groups of virtual machines
  - Host - Runs the Hypervisor
  - VM - virtual machine or guest machine
  - Linode - VM as a product (what we sell)
  - Network - Network topology
  - System Classes - collection of servers that make up a system
  - Terraform - tool to orchestrate deployment of VMs
  - Ansible - tool to configure VMs
  - Salt (saltstack) - tool to configure VMs

Building an Internal Testcloud: salt-pillar

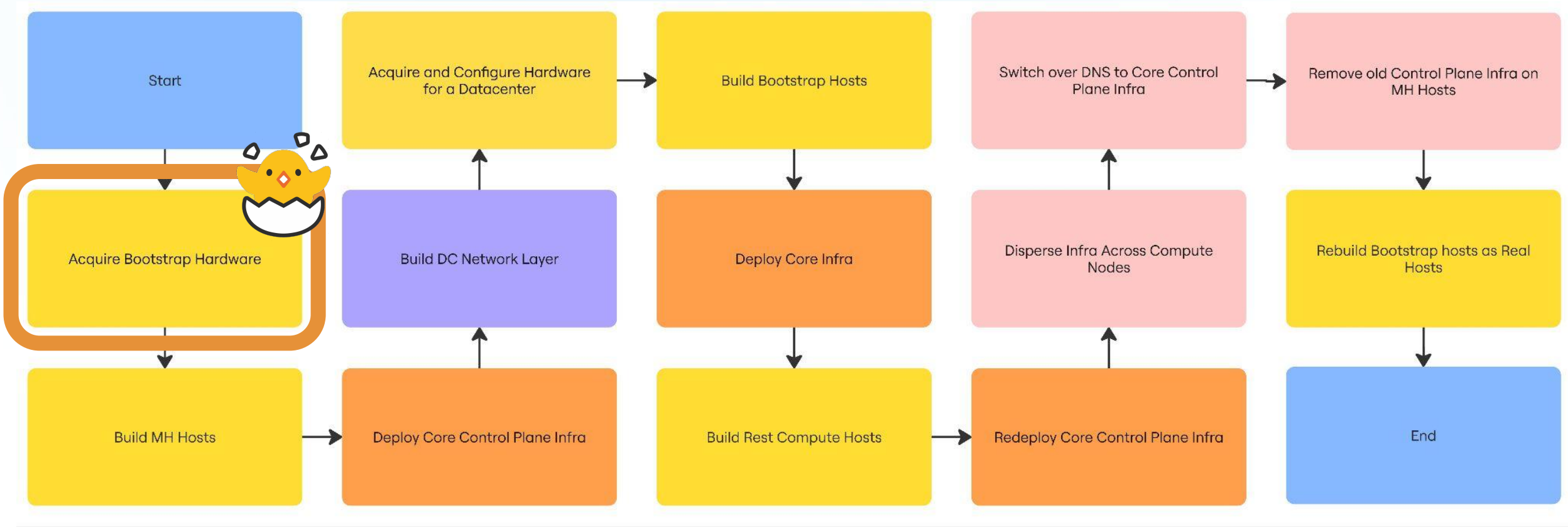
```
## packages.sls
bind:
    package-name: bind
    version: 9.16.1-0.1
    port: 53
    listen-on: any

## override.sls
bind:
    version: latest
```

```
$ salt-call pillar.get bind
local:
    ----------
    bind:
        ----------
        listen-on:
            any
        package-name:
            bind
        port:
            53
        version:
            latest
```

Building an Internal Testcloud: salt-states

```
bind_package:
  pkg.installed:
    - name: {{ salt['pillar.get']('bind:package-name', 'bind9') }}
    - version: {{ salt['pillar.get']('bind:version', 'latest') }}
```

Building an Internal Testcloud: Double Bootstrap



Building an Internal Testcloud

Lab Network

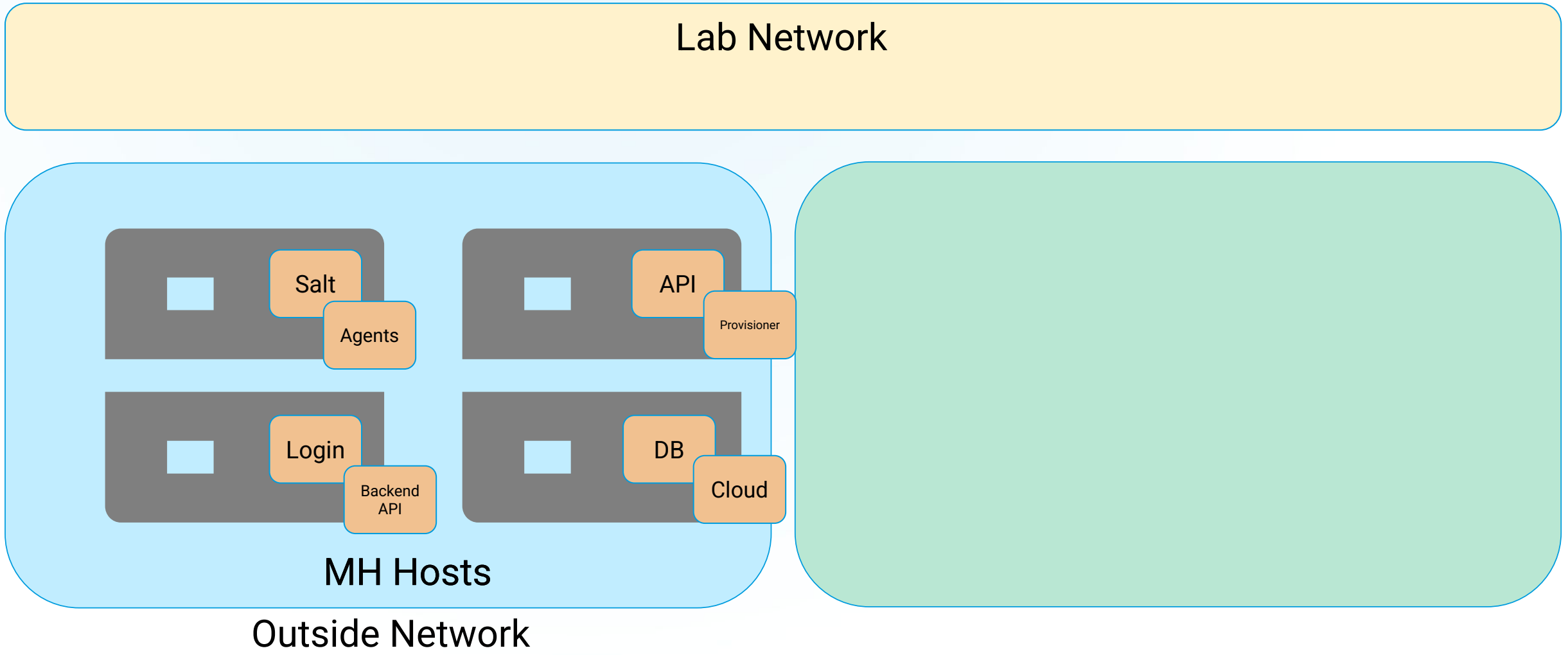
MH Hosts

Outside Network

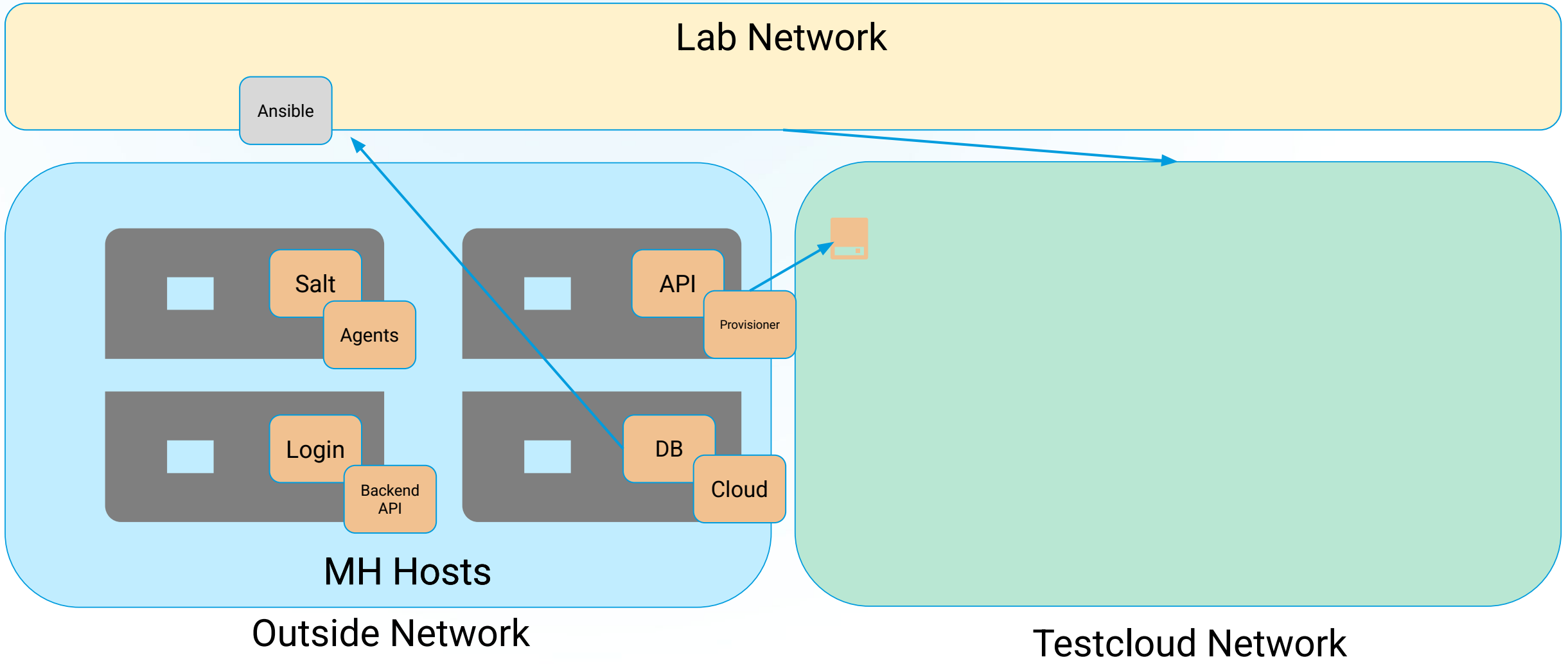
Building an Internal Testcloud: testcloud-ansible

- Targeted a VM by regex and static IP address that was defined in our inventories
  - Libvirt VM or Vagrant VM
- Core system classes Salt and Vault
  - Vault: added environments where for each secret to create placeholders
  - Salt: not good at installing salt
  - DB was first
- Once we had these, all other systems we could “just highstate” (if the vault and db information were correct, which took time)

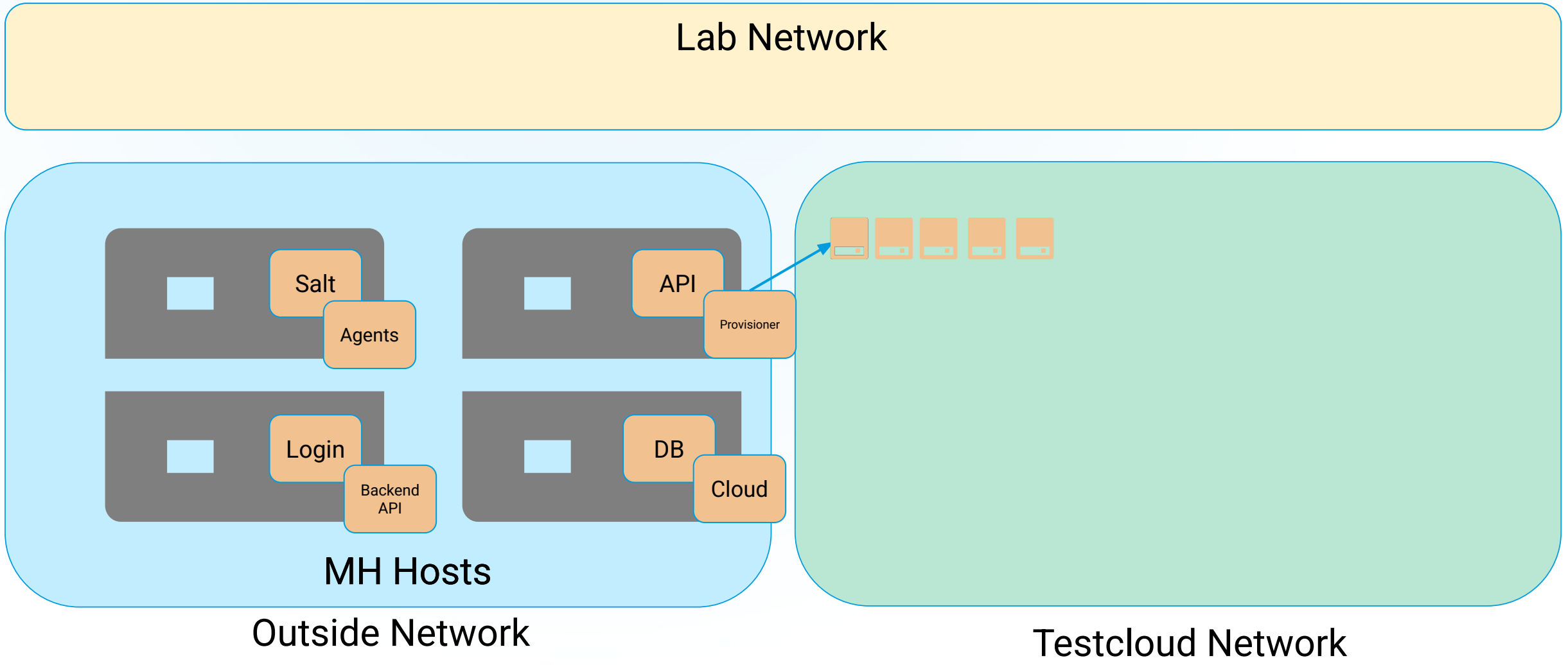
Building an Internal Testcloud



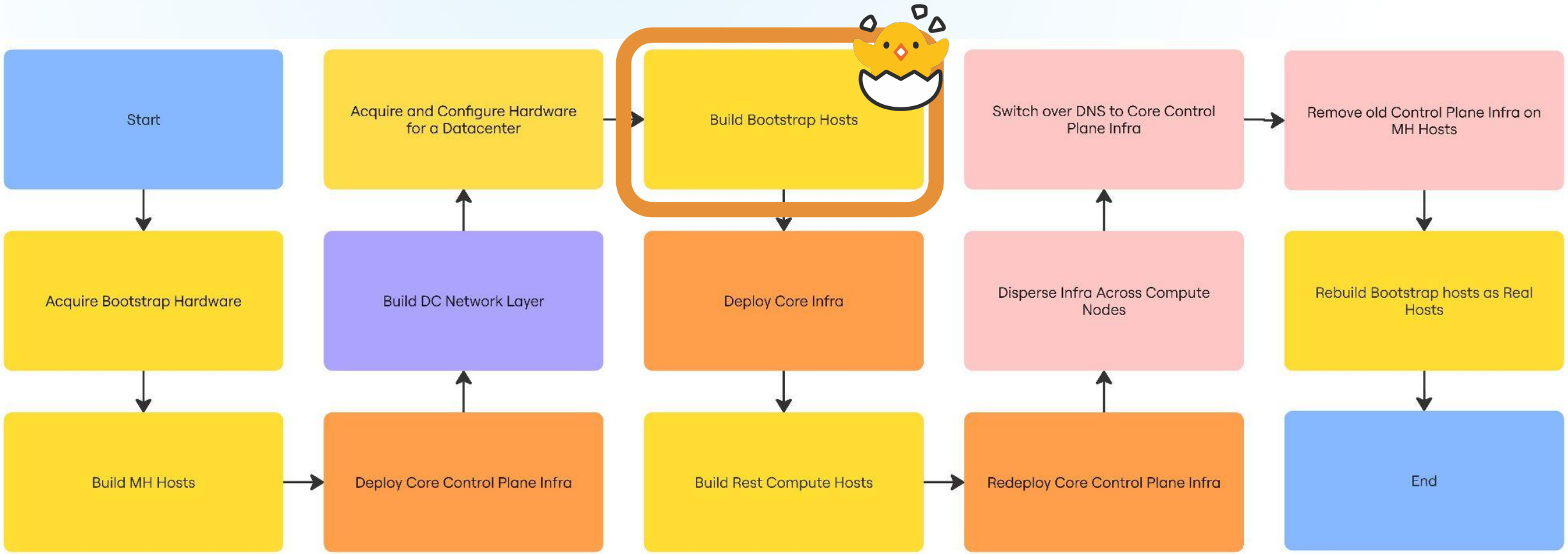
Building an Internal Testcloud



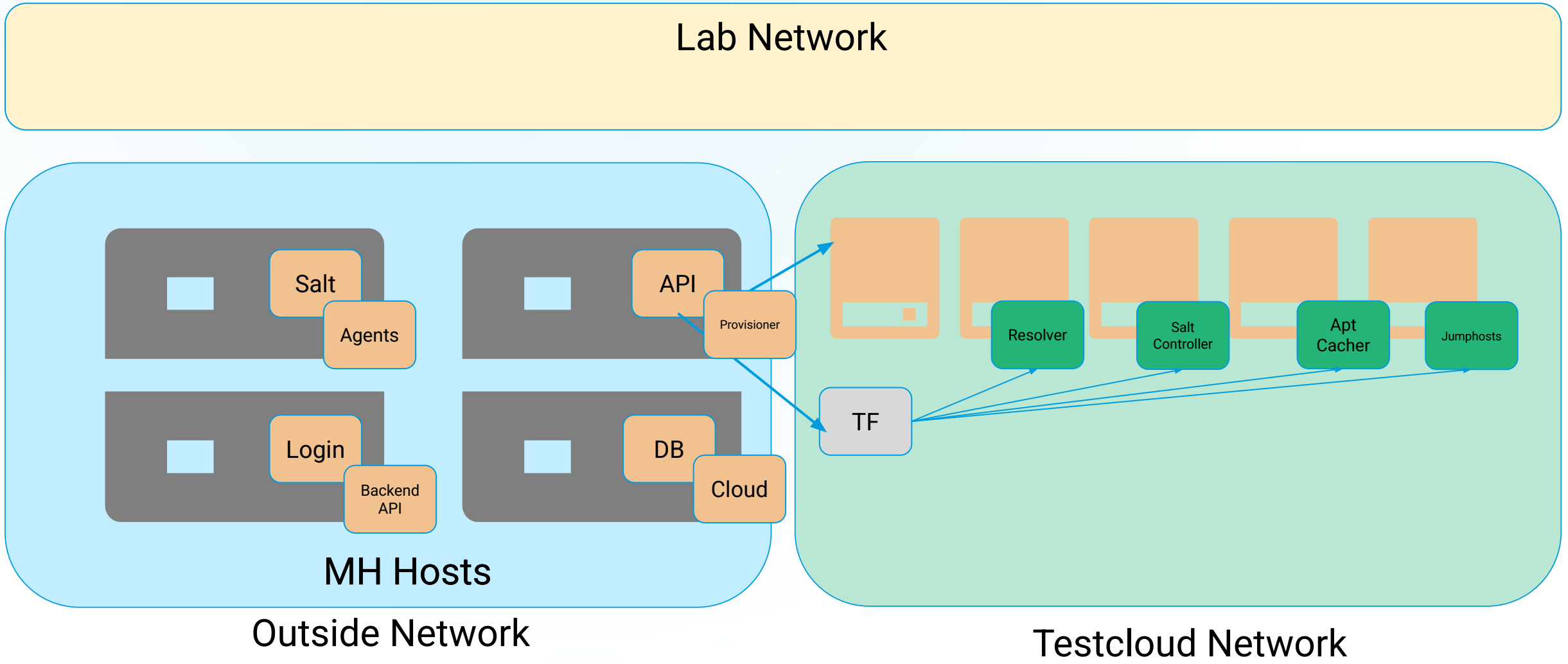
Building an Internal Testcloud



Building an Internal Testcloud



Building an Internal Testcloud

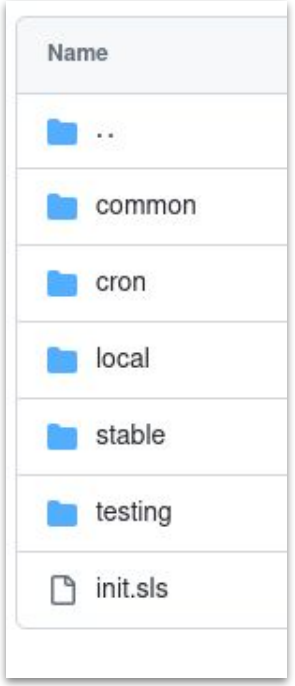


Building an Internal Testcloud: terraform-module-infra

- Internal client of our Linode Terraform Provider (TMI)
- Custom Modules that enforces standards
  - Creates DNS via Edge DNS API / Provider
  - Static networking where needed
  - Custom human-friendly DNS
  - Installation of Salt and grains for DC information
  - Baseline SSH configs and other sec hardening
  - Leverage Linode Tags for things like monitoring, etc
  - Consistent naming convention

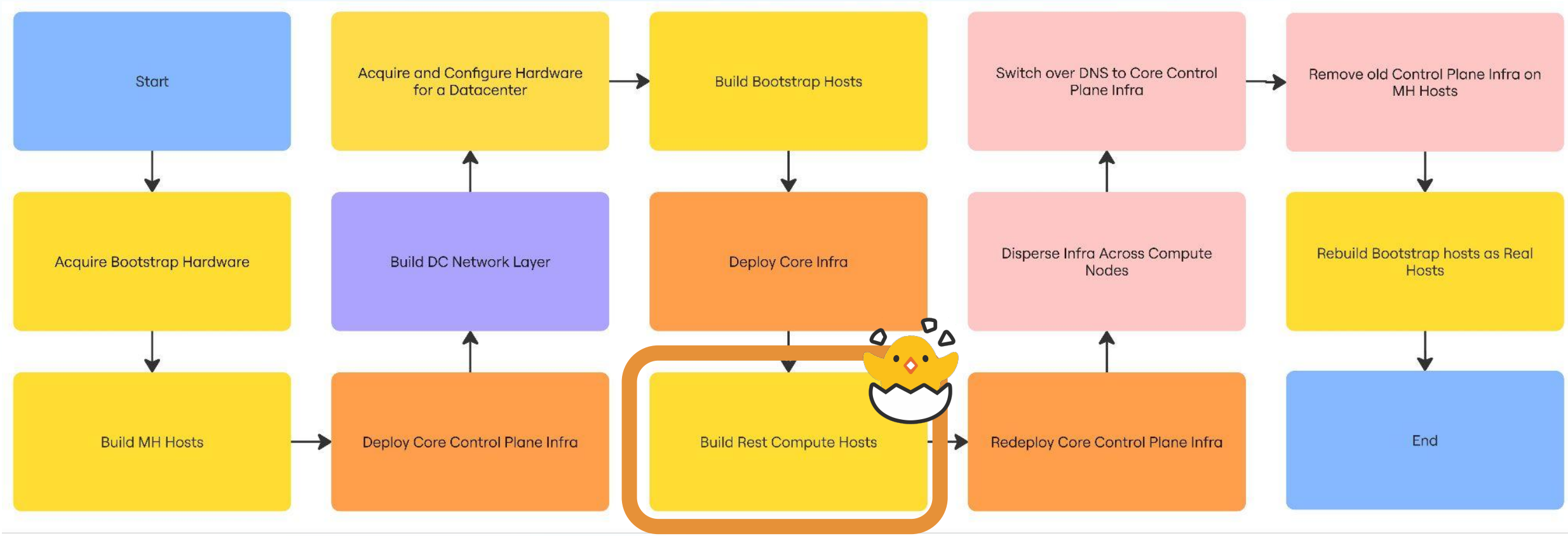
Building an Internal Testcloud: salt-pillar

- How to make our configs DRY but keep things isolated?
  - Driven by grains set by an internal module that sets DC specific information
  - Layered pillar (prod as the base + layered lower env on top)
    - separate Vault but same paths or pillar rendering issues
  - Unexpected pillar merging issues
  - Issues like if the key is present but empty, salt will fail or other
  - Lists vs. dictionaries merging
  - Confusing order of operations for deploying net-new systems to Testcloud

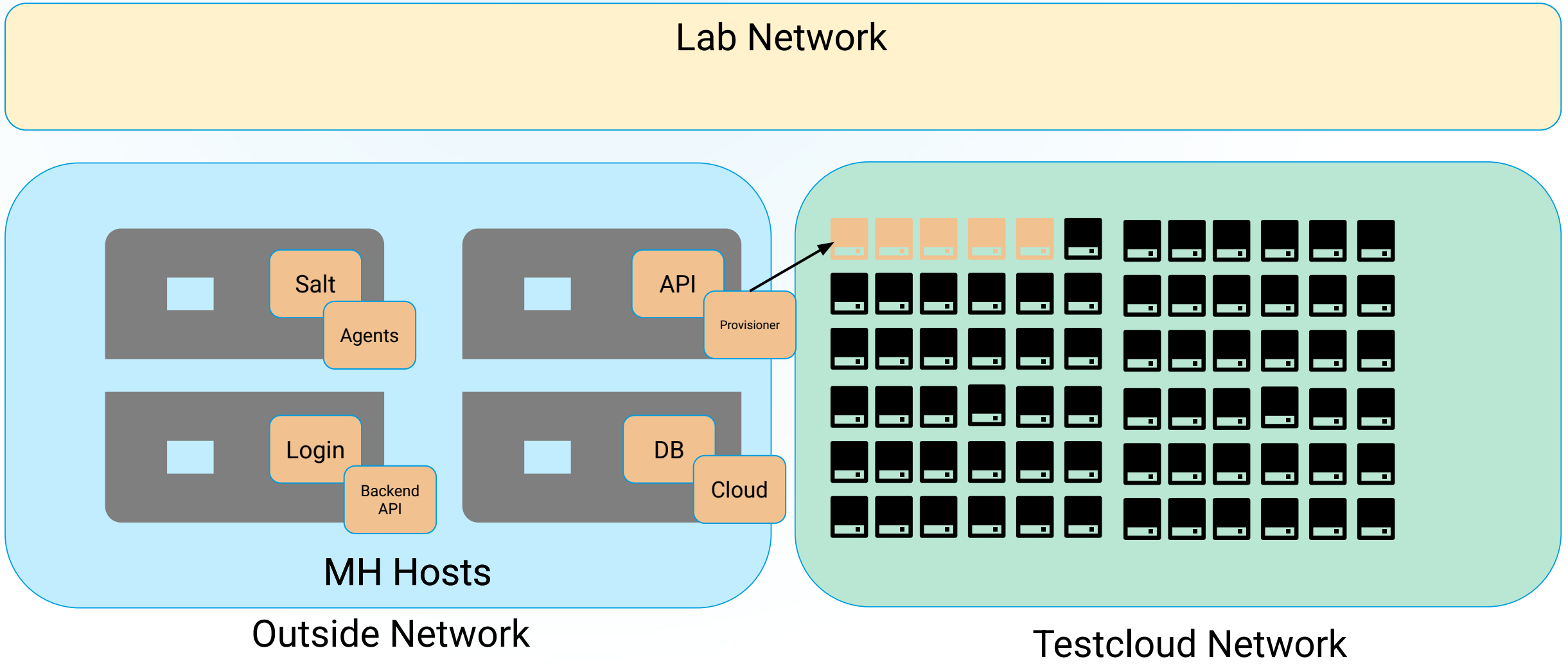



```
{% set environment = grains.get('linode', {}).get('environment', 'stable') %}

include:
  - override.login.common
  - override.login.{{ environment }}
```

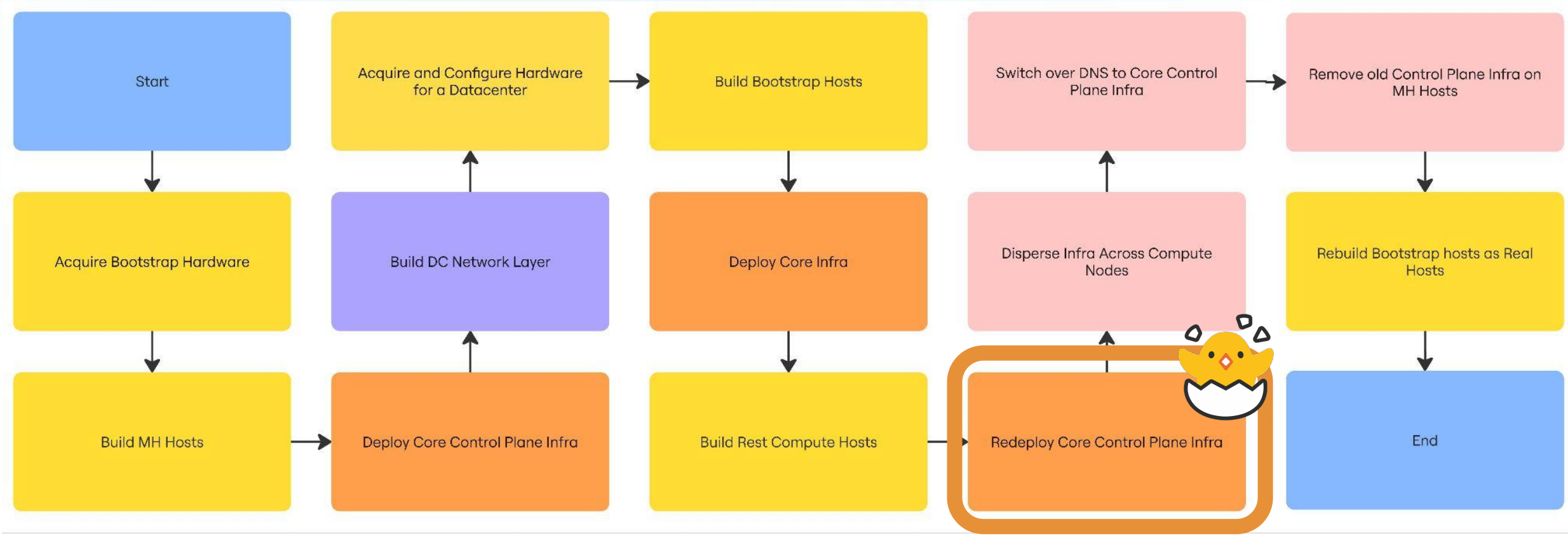
Building an Internal Testcloud



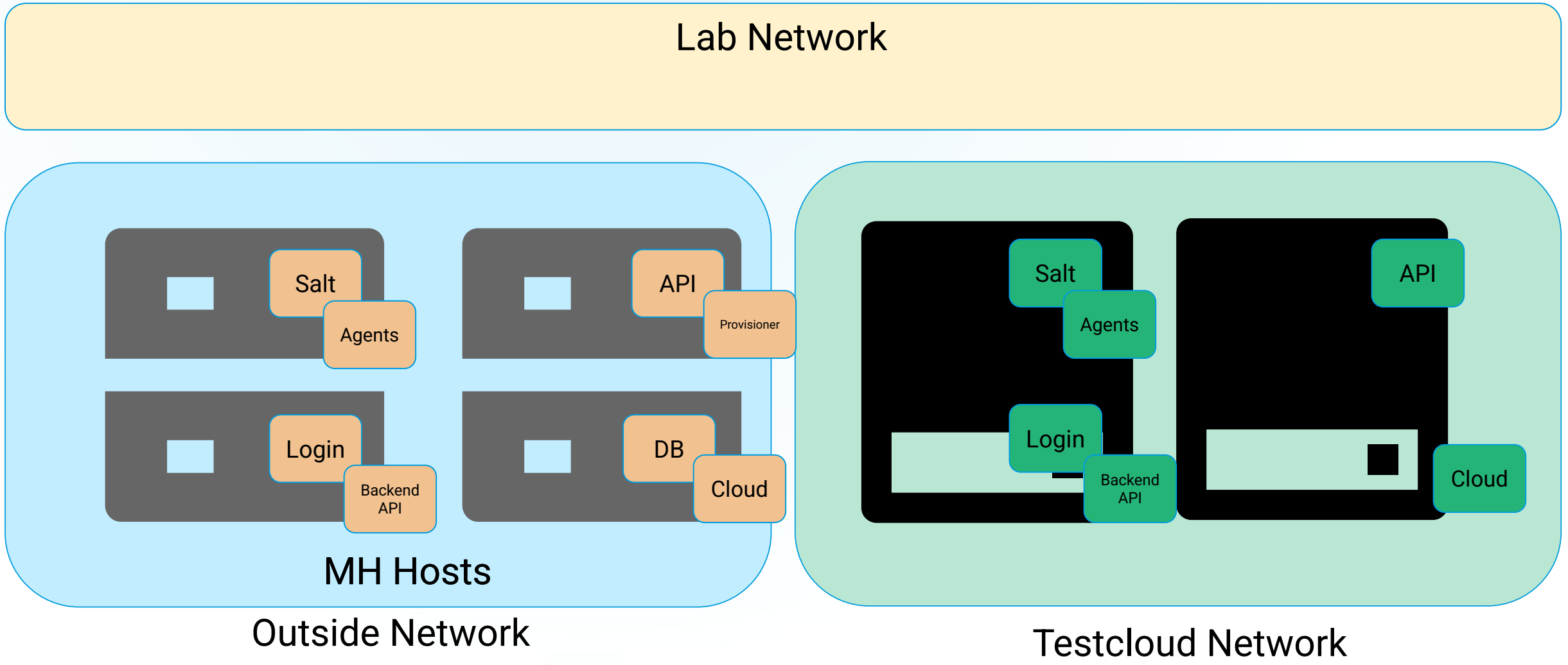
Building an Internal Testcloud



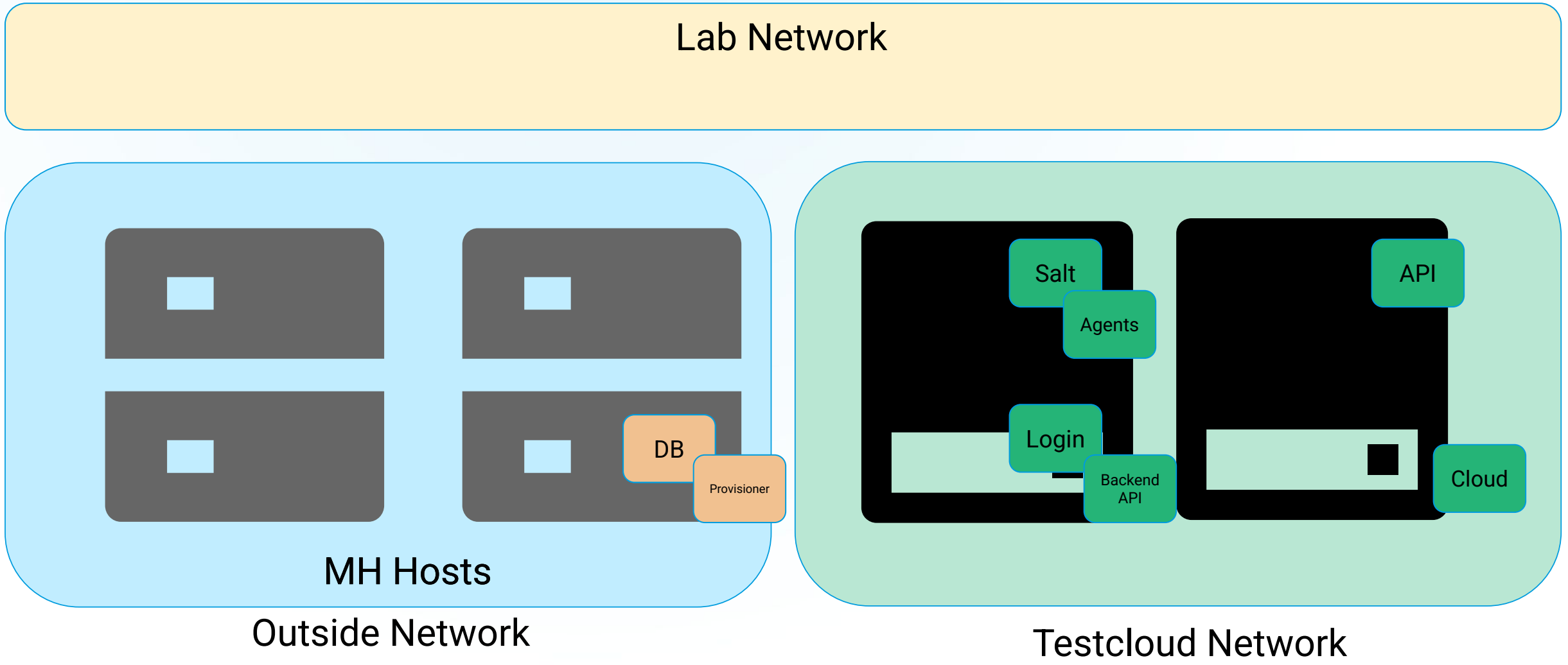
Building an Internal Testcloud



Building an Internal Testcloud

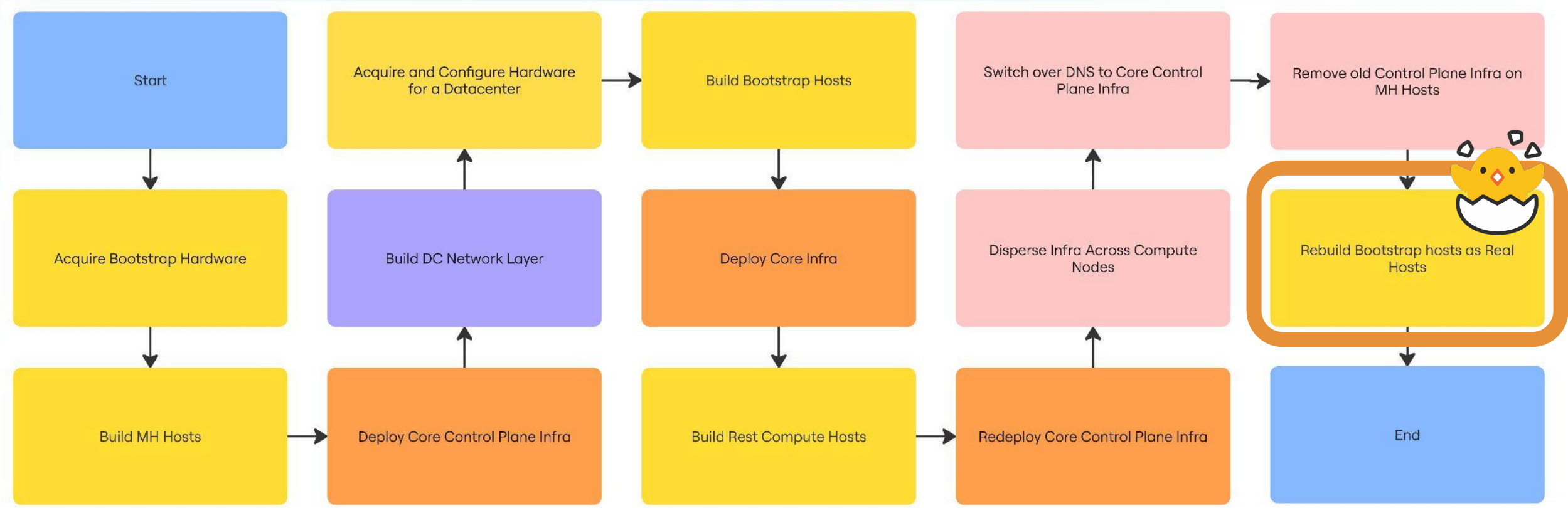


Building an Internal Testcloud

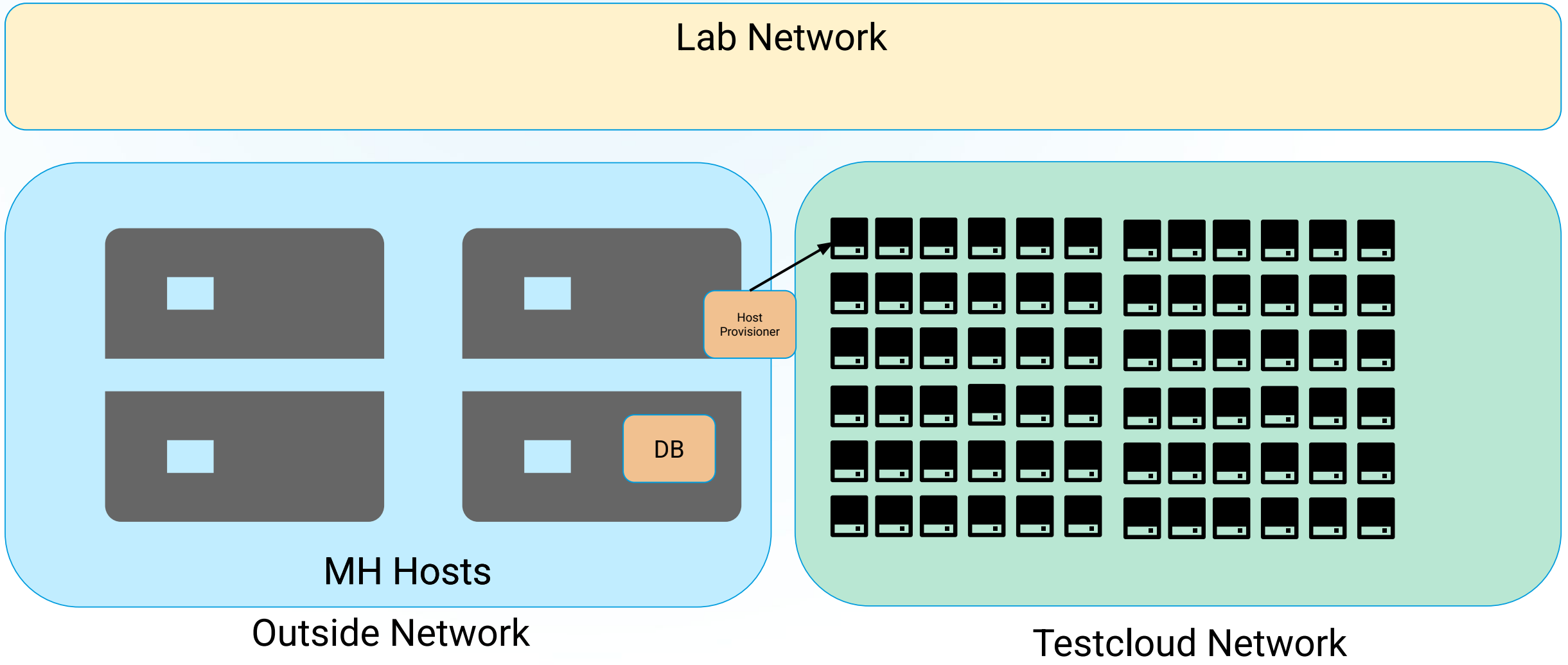


Double Bootstrap Method

Building an Internal Testcloud



Building an Internal Testcloud



Why so complicated?

Overview: Requirements

- …
- **Need to maintain parity**

The Journey

- Overview
- Building an Internal Testcloud
- **First Customers**
- Wins and Lessons Learned

First Customers

- Our first customers were the compute team, they are responsible for building our main stack
- Each member of the team was assigned a team to help them onboard
- Setup Customer accounts by Department, using Terraform to manage with the Linode provider
  - Allows us to track usage per department
  - Allows us to allocate Infra to different teams
  - Connects to “reservation” service for tracking, using and cleaning of Compute Hosts

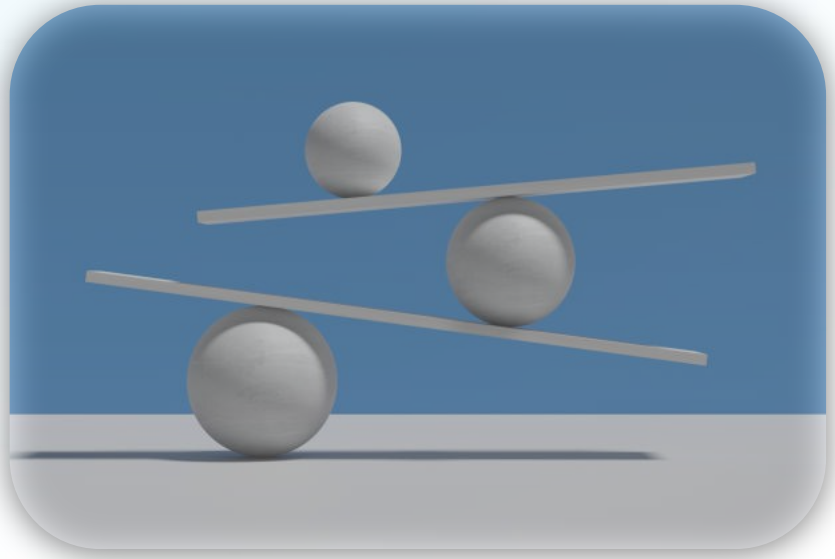
Dedicated Team

Team structure was set up across 3 geos: NA, EMEA, and APAC

- We have a “firechief” rotation to assist customers and help onboarding and other tickets and provide help during all geo timezones
- We try to make things as self service as possible, and encourage teams to assign code owners
- We automate toil but there is still a pretty heavy volume of requests.

Maintaining Stability

- Dedicated team
- Alerting same as Prod
- Have incidents when things are broken
- Culture and naming of “stable”



Value Add

- Testcloud is now in the release path of almost every product
- We caught many important bugs before they made their way into our production environment
- No need to worry about things being on the internet and tests becoming compromised – it’s a huge win!
- We can test things we weren’t able to before
- Chaos testing, performance testing and seeing the upper bounds of our applications

The Journey

- Overview
- Building an Internal Testcloud
- First Customers
- **Wins and Lessons Learned**

Wins and Lessons Learned

- Platform as an internal private cloud
- To date we have over 500+ customers
- Getting to 100% parity is nearly impossible, but we are very close

Wins and Lessons Learned

The biggest surprises were

- Difficulty of getting prioritized engagement from other teams
- The integration from our Previous Test Datacenter provided and folks expected to be part of stable not being present
- The tech debt we took on from hard forking production repos really was underestimated.
- What people built on the platform
  - We have seen a ton of innovation in terms of expanding our tooling, especially our local dev tooling

Takeaways

- When building an internal Pre-prod platform with circular dependencies, use the “double bootstrap” approach
  - Build Thing1, use Thing1 to build Thing2, replace Thing1 with Thing2, delete Thing1
- Don’t underestimate the power of interpretation
- Front-load scaling tasks as they come up (e.g. hard coded domain, ip range, etc)
- Resist the fork and push for changes upstream and manage override configs for your specific environments.

THANK YOU!

Special Thanks to the Core Team

- Adam Twardosz
- Aditya Jambhalikar
- Alan Laird
- Ananthu Sajithkumar
- Chaitanya Kirkase
- Chris Row
- Craig Rotbert
- David Bennett
- Dean Carpenter
- Dima Kondratskyi
- Medha A
- Pat Thiel
- Roshan Baladhanvi
- Samuel Chapler
- Siddhi Kadam
- Steven Jacobs
- Swapnil Srivastawa
- Tom Ralph
- Twisha Das
- Wasim MD