Atlassian Hybrid Cloud/On-Premises Software Delivery

and the journey to 300,000 applications in the cloud

GEORGE BARNETT · SAAS PLATFORM ARCHITECT · ATLASSIAN

About me

- I’ve been building infrastructure for companies since the late 90’s
- Atlassian for 7 years
- Last 5 years building the platform underlying Atlassian Cloud

What are we talking about?

- What we've learned while providing both On-Premises and SaaS offerings to our customers
- How to use infrastructure to assist developer teams while they adjust to new delivery channels
- Tips for running a massive number of apps
- Ideas for your infrastructure

A long time ago in a country far away...

DOWNLOAD

JIRA
The brilliantly simple, incredibly powerful way to track and manage issues.

DOWNLOAD

Confluence
The most advanced enterprise wiki, Confluence helps teams to collaborate and share knowledge.

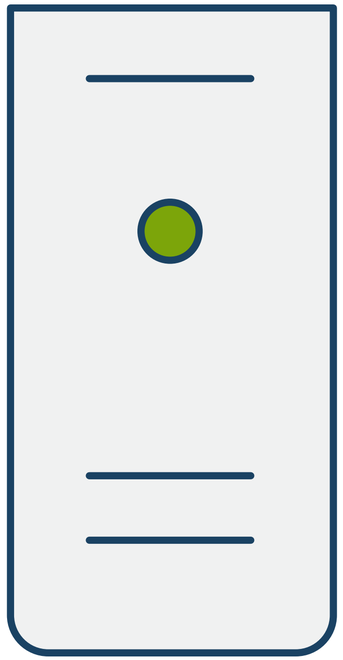
DOWNLOAD

Bamboo
No ordinary continuous integration server, Bamboo brings you build results, metrics, trends, patterns, links and more.

DOWNLOAD

Download Native

- Many versions, run on customer infrastructure
- Features into Major releases
- Bugfixes into updates & backports
- Extended release cycle - months
- Usually a monolith to ease deployment
- Config often from os environment (hostname, timezone,..)



Cloud Native

- Services continuously deployed
- Feature delivery is incremental and behind a feature flag
- Bug fixes released when ready
- Very short release cycle - hours
- Service discovery is normal

Iteration #1.
JIRA Studio

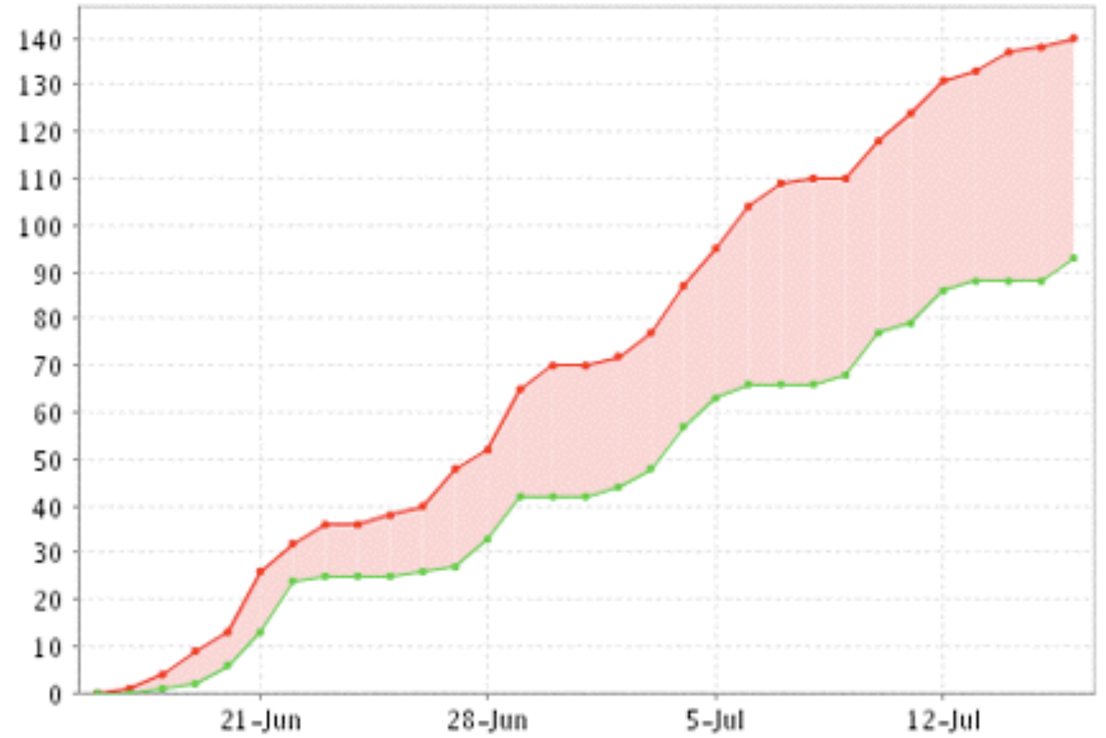
> “Rewriting code from scratch is the single worst strategic mistake that any company can make.”

JOEL ON SOFTWARE - THINGS YOU SHOULD NEVER DO, PART I

Atlassian Projects

Manage Portal

Created vs Resolved Issues: **iEat Server** (IEAT) more detail >>



Issues: **140** created and **93** resolved
Period: last **30** days (grouped **daily**)
View detailed data table >>

Activity Stream: **All Projects**

July 2008

Douglas Butler committed changeset 10 to the IEAT project saying:
This is a partial fix of IEAT-1

Douglas Butler committed changeset 9 to the IEAT project saying:
This is an incremental fix of IEAT-1

System Administrator and **Tim Moore** edited Marketing Requirements making:

System Administrator: 1 change

System Administrator: 1 change

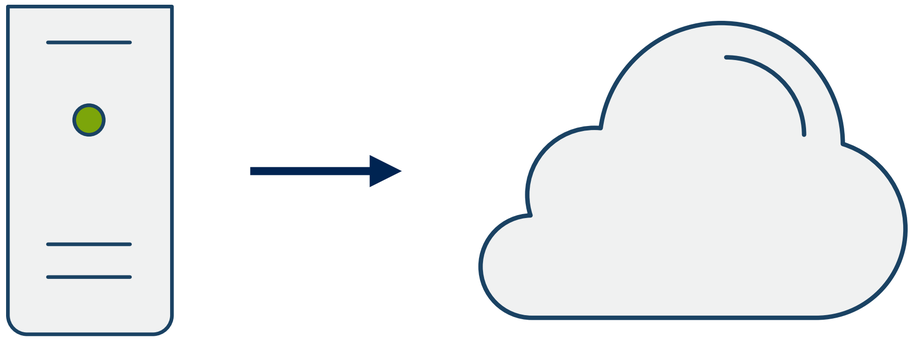
Tim Moore: 1 change

June 2008

Douglas Butler committed changeset 6 to the IEAT project saying:
Partial fix of [IEAT-2]

Download Native in Cloud

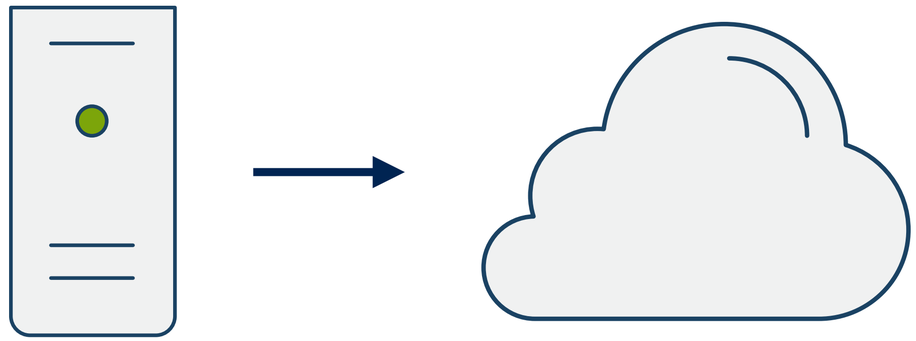
- Overlay integration on top of existing products
- 1 VM per instance, 3rd party hosting
- Release every ~3 months
  → Wait for 4 products on various cycles
  → Integrate all the things (8w)
  → 3rd party bakes VMs (2w)
  → Upgrades to Prod (2w)



Our initial solution ignored the ergonomics of the new delivery channel

Pain points

- Time pressure on product teams
  - Unintended downstream dependency on releases
- 3rd party hosting provider
  - Poor visibility into production setup
  - High per VM cost with long HW lead time
- Extended release cycle
  - Customers always behind in features
  - Bugfix MTTR is long
- Individual configs add complexity for Support teams



High performing dev teams can become low performing teams if switched too fast

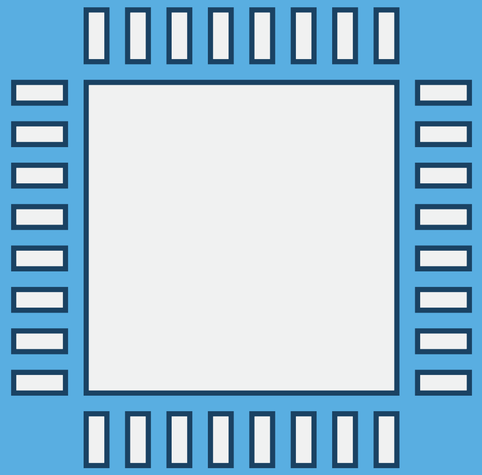
Reduce the surface area of change

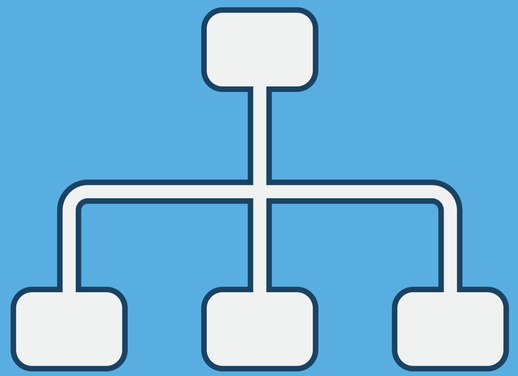
Iteration #2.
Atlassian OnDemand

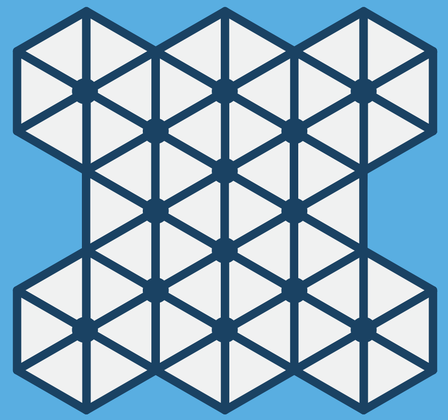
Hide the scale from developers & reuse existing app architecture.

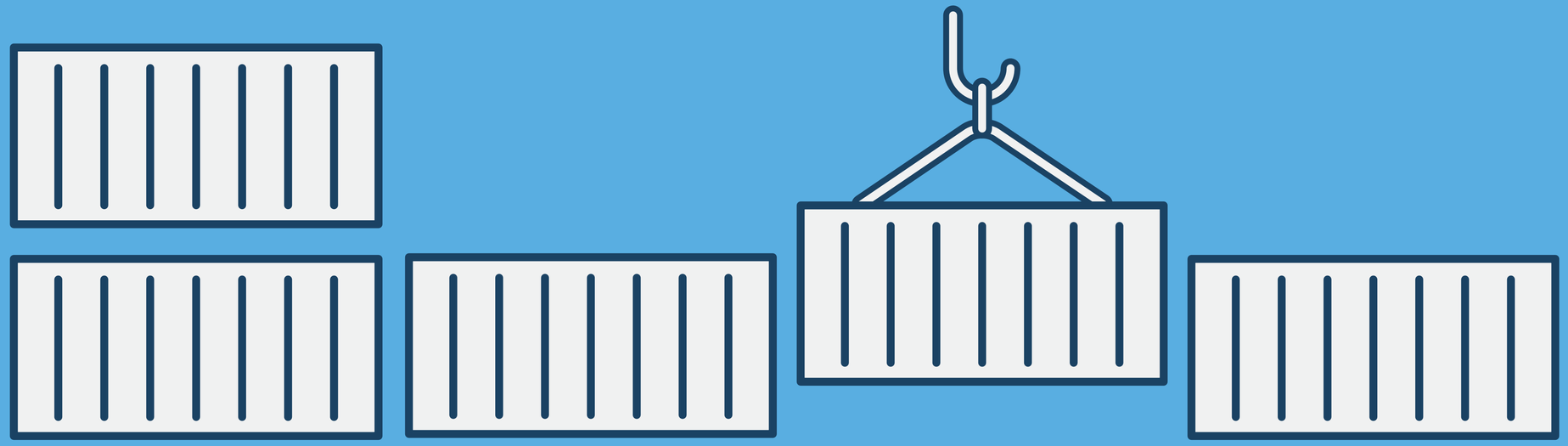
Use infrastructure to wrap multi tenancy around the application

Begin by creating an easy to deploy hardware platform

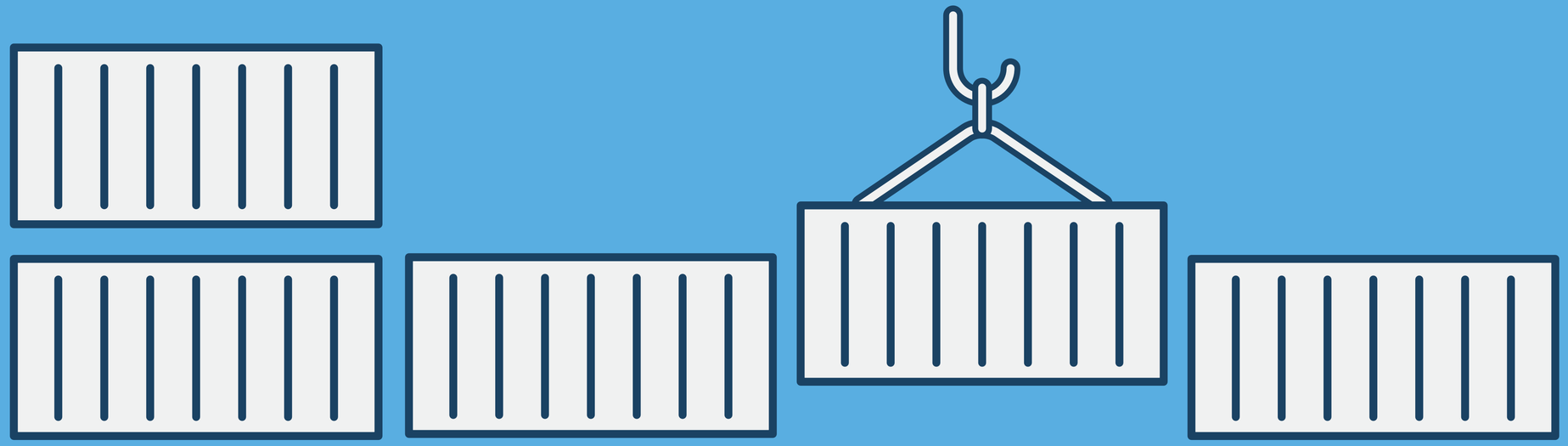


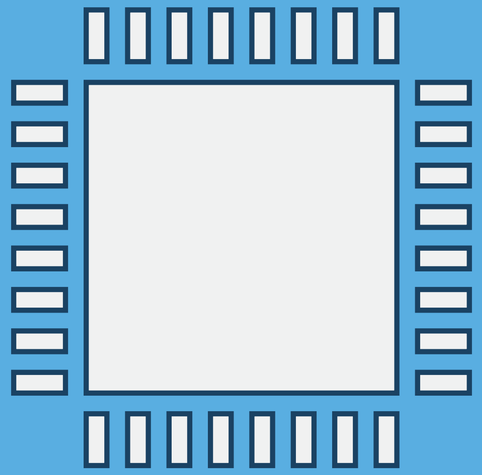


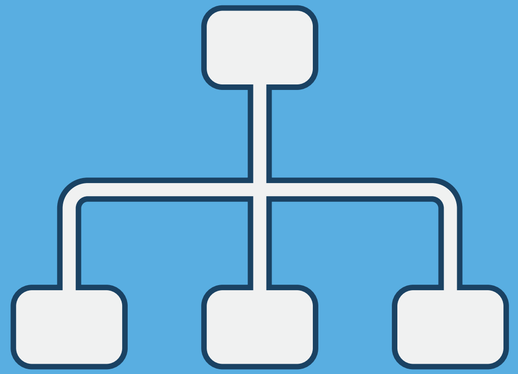


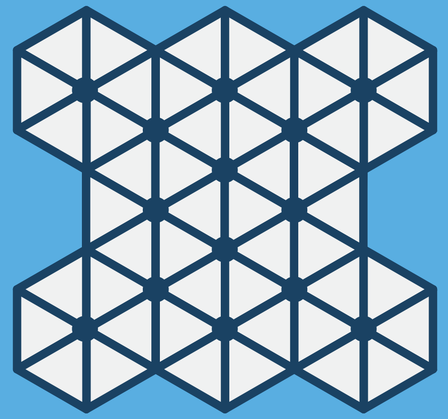


Use containers to wrap applications inside a standard environment

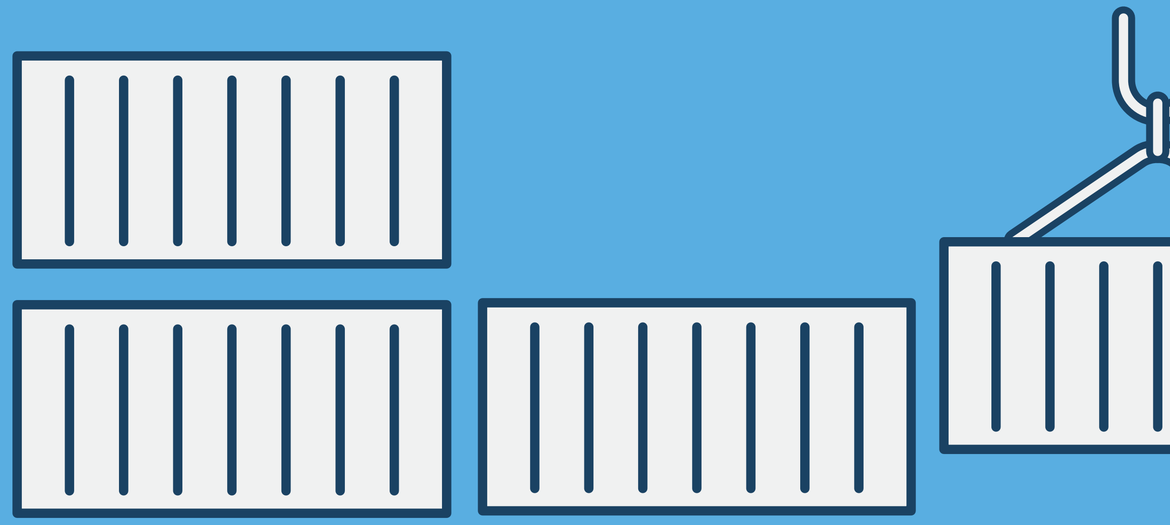


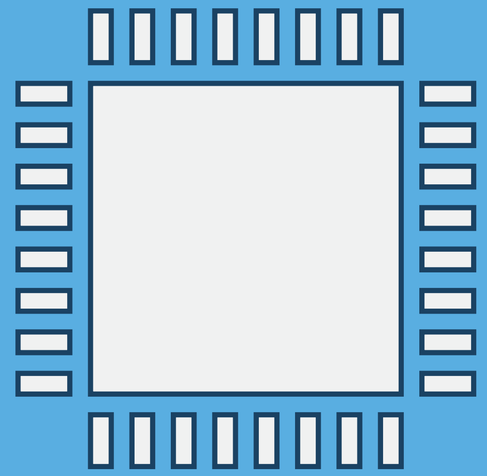


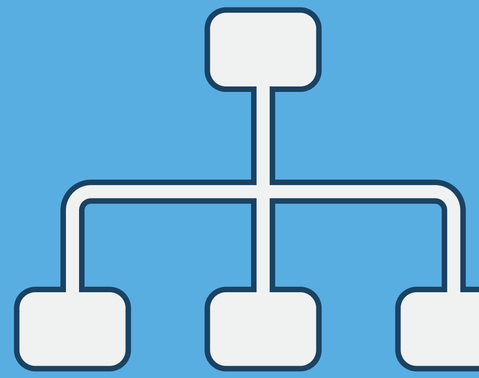




Enforce strict ‘top half’ / ‘bottom half’ separation to decouple dev and infrastructure







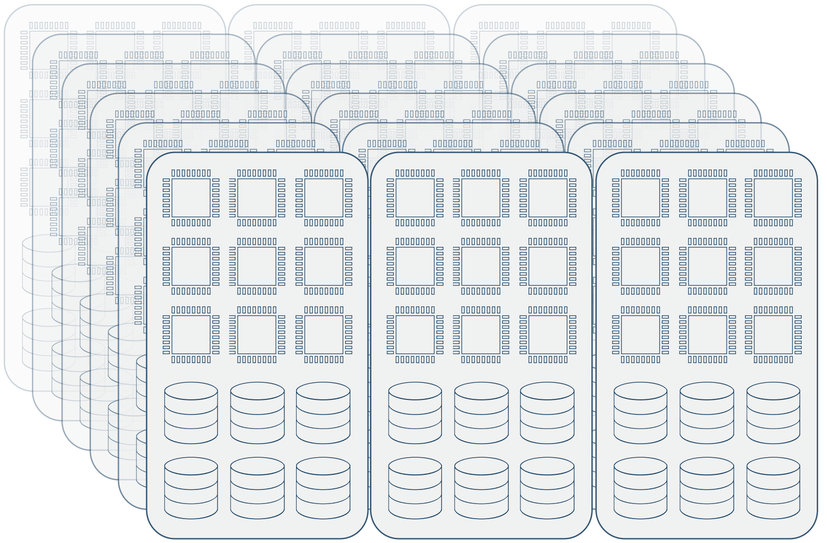
Create services that enable repeatable application configuration

Templates

Containers
DNS
Monitoring
Proxies
Cloud Peering

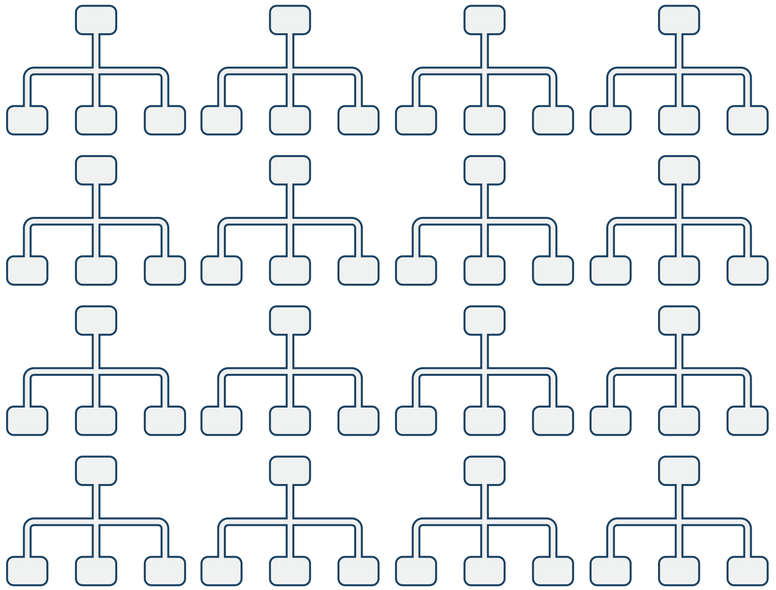
Templated Hardware

- Preconfigured build from systems integrator - no rack & stack
- 2 Server SKUs - Compute & Storage to reduce complexity
- Deliver; Plug-in; Boot up
  - Infrastructure team can spend cycles on platform improvements
  - 1 rack minimum deployment unit
  - Customers are sharded by rack



Templated DCs

- Pre-baked configuration template contains a ‘whole DC’
  - IP supernet and DNS namespace
  - Delivered racks installed into ‘slots’ in the template
  - Optimise for time to deploy
  - Pre-baked Compute and Storage hardware node OS images
- Config management adds zone wide services
  - LDAP, DNS resolvers, Provisioning APIs
  - Per rack management services



Templates

Containers

DNS

Monitoring

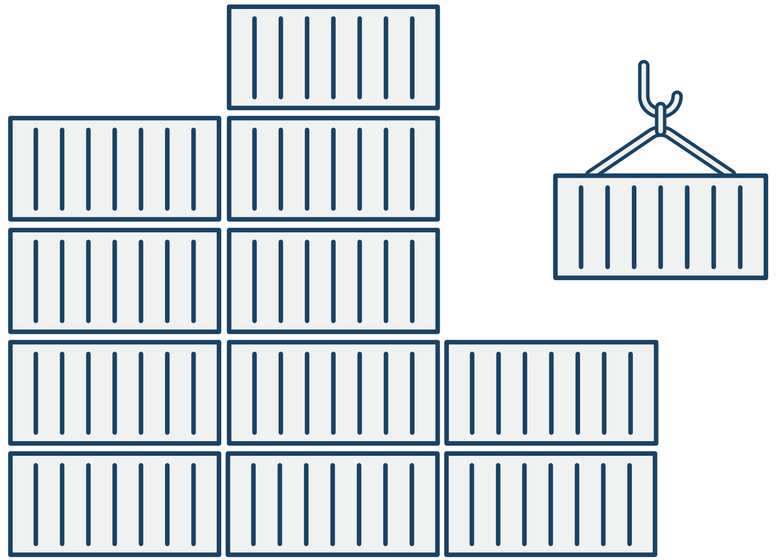
Proxies

Cloud Peering

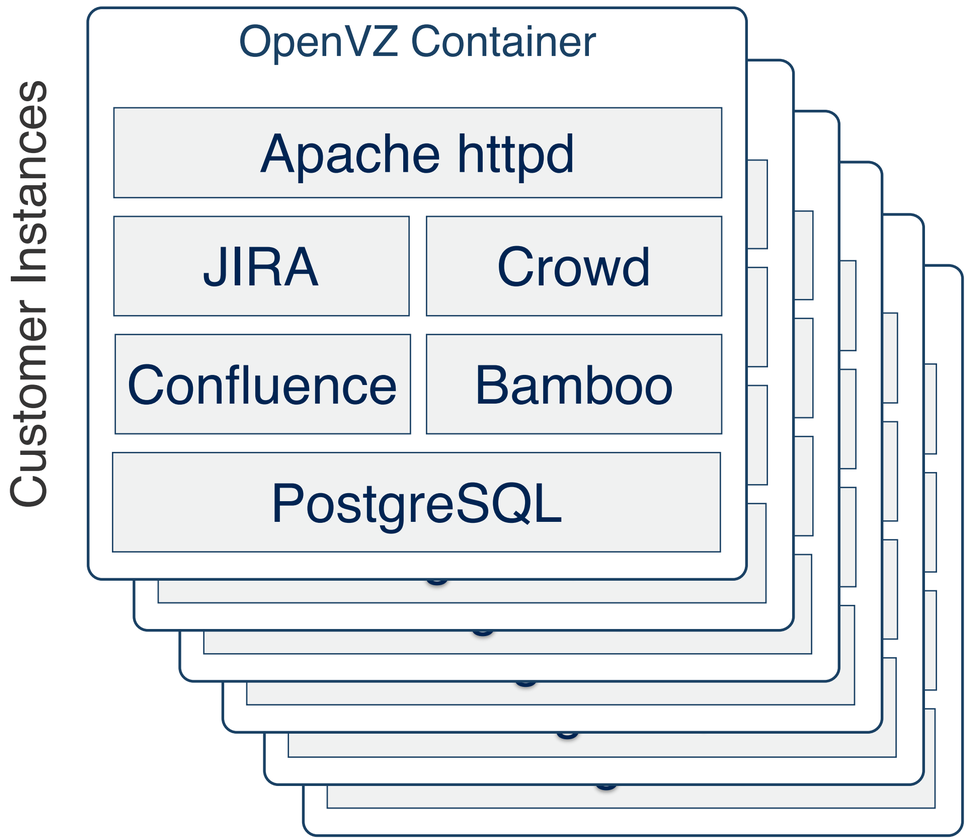
Containers

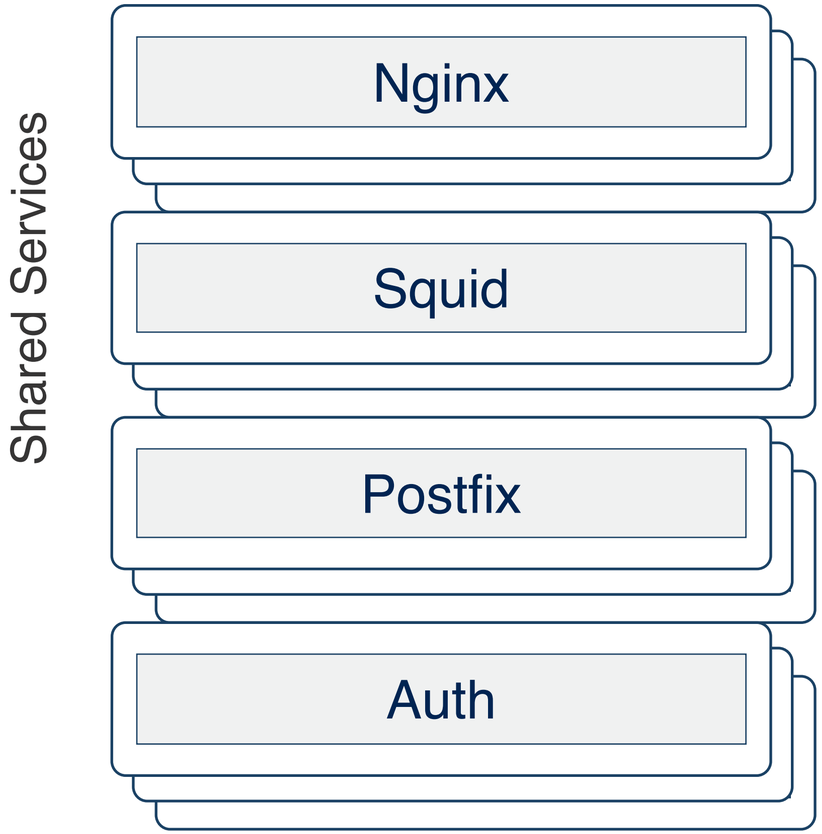
Runtime

- Containers for everything that isn’t required to run the platform
- OpenVZ
  - Very stable platform with an excellent feature set - CRIU, ploop, isolation, resource accounting
  - Customisations allow fast provisioning.
  - Lightweight and saves memory.
  - Simulate a full VM - hostname, networking



Containers


Customer Instances
OpenVZ Container
Apache httpd
JIRA
Crowd
Confluence
Bamboo
PostgreSQL


Shared Services
Nginx
Squid
Postfix
Auth

Containers

- By convention, containers get 3 mounts

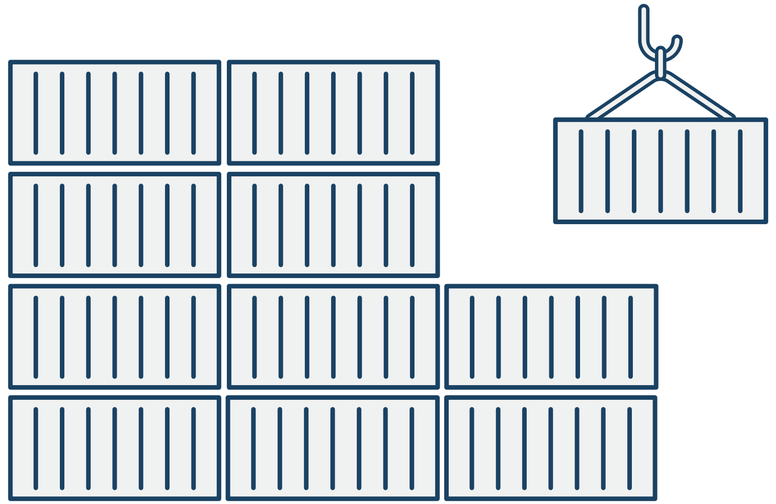
read only

/	Userland Prototype. Identical everywhere.
/sw	Software Repo. Identical per environment.

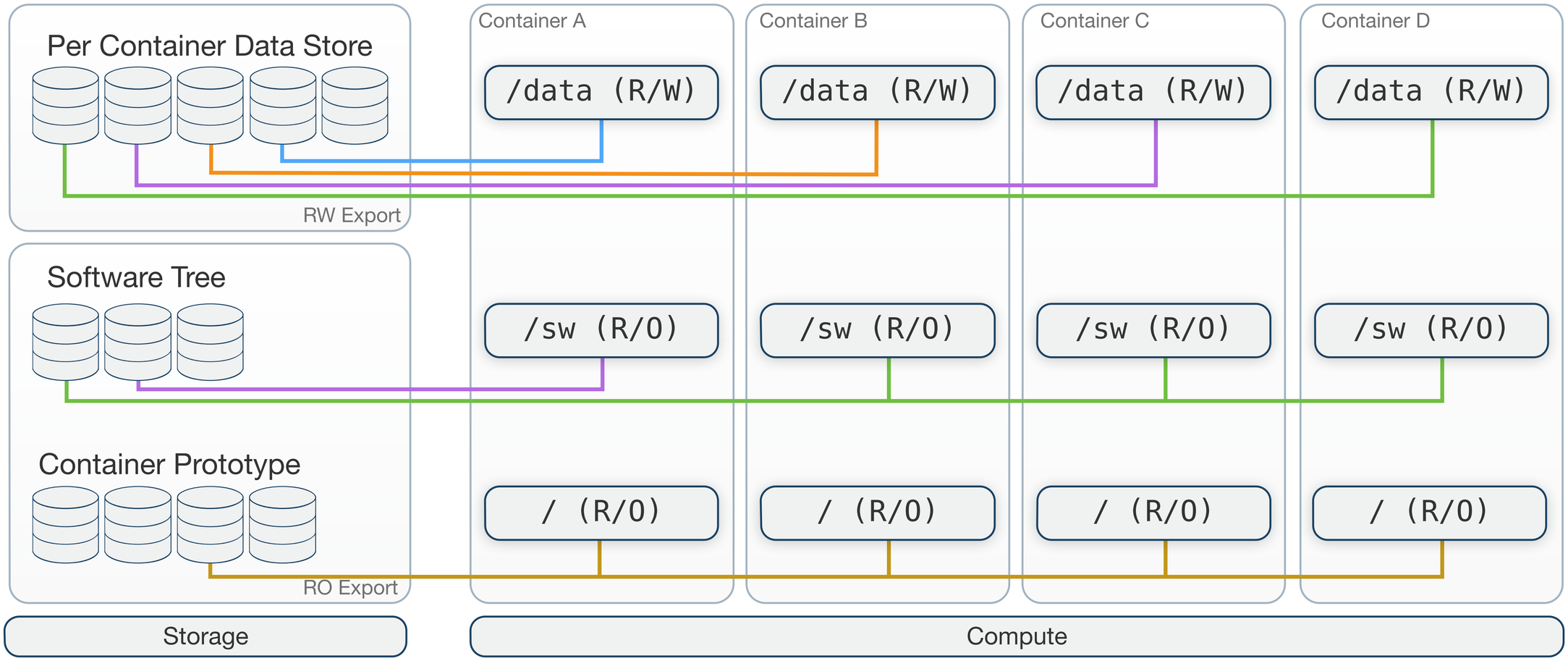
read/write

/data	Per container persistent storage.

- Read only mounts:
  - Most of container is immutable
  - Bind mounts share underlying block cache

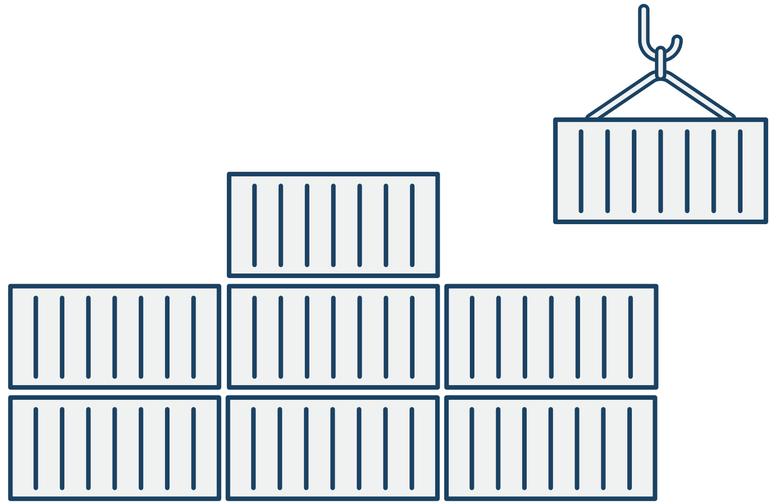


Containers


Per Container Data Store
RW Export
Software Tree
Container Prototype
RO Export
Container A
Container B
Container C
Container D
/data (R/W)
/data (R/W)
/data (R/W)
/data (R/W)
/sw (R/0)
/sw (R/0)
/sw (R/0)
/sw (R/0)
/ (R/0)
/ (R/0)
/ (R/0)
/ (R/0)
Storage
Compute

Containers

- Multiple processes in container
- Lightweight simulation of full VMs
  - Allows use of many cluster tools
  - Minimum set of base services (cron, syslog, etc)
  - Everything else is under daemontools.
  - Service management via fab, puppet (depends on team).



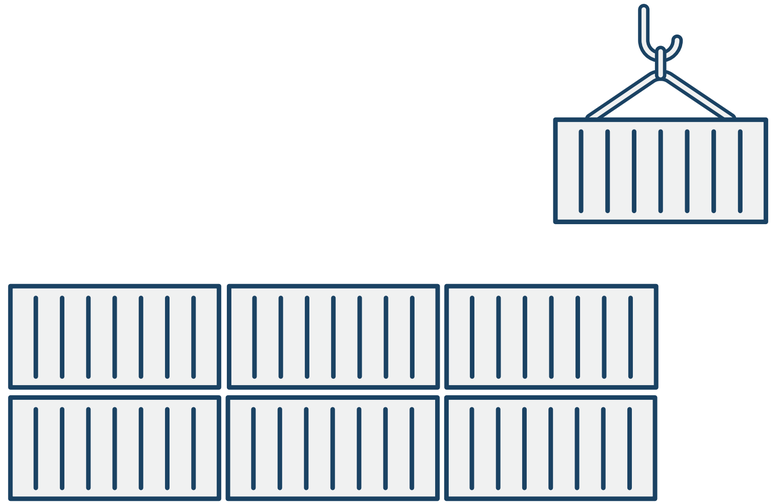
Containers

```
PID TTY      STAT   TIME COMMAND
   1 ?        Ss     0:03 init [3]
1302 ?        Ss     0:00 syslogd -m 0
1310 ?        Ss     0:00 crond
1318 ?        Ss     0:00 incrond
1482 ?        Ss     0:00 /bin/sh /usr/bin/svscanboot
1484 ?        S      0:01  \_ svscan /data/service
1490 ?        S      0:00  |   \_ supervise sys_collectd
1505 ?        Sl     0:04  |   |   \_ /usr/sbin/collectd -f -C /sw/monitoring/collectd/conf/cloud.conf
1502 ?        S      0:00  |   \_ supervise sys_sshd
1519 ?        S      0:00  |   |   \_ /usr/sbin/sshd -e -D -f /sw/sshd/config/sshd_config
1492 ?        S      0:00  |   \_ supervise sys_postgresql
1516 ?        S      0:00  |   |   \_ /usr/bin/postgres -c config_file=...
1496 ?        S      0:00  |   \_ supervise j2ee_confluence
3205 ?        Sl     9:42  |   |   \_ /sw/java/sdk/Sun/jdk1.8.0/bin/java ...
1498 ?        S      0:00  |   \_ supervise j2ee_jira
3206 ?        Sl    10:10  |       \_ /sw/java/sdk/Sun/jdk1.8.0/bin/java ...
1485 ?        S      0:00  \_ logger -p daemon.info
```

Containers

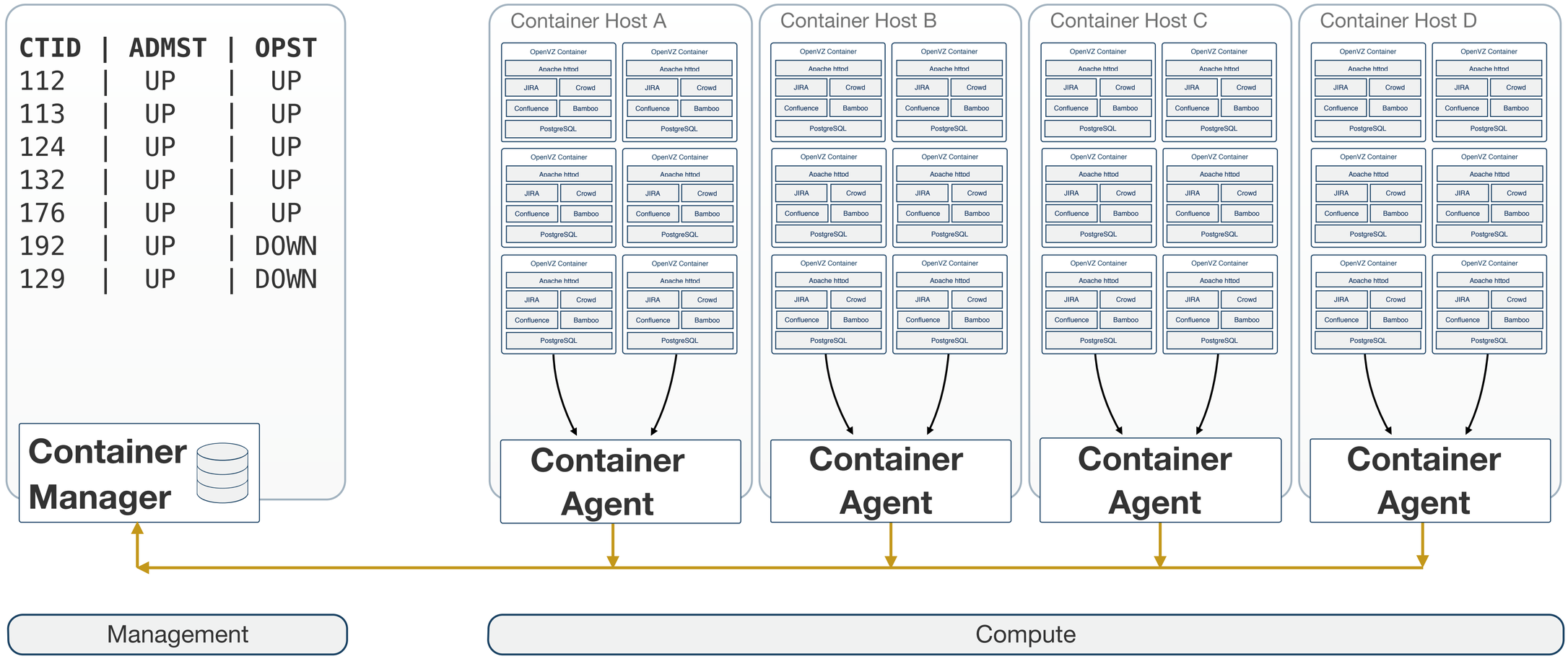
Networking

- Each shard (rack) has 1 x /16
- Each container gets 1 IP address
  - Less friction for services on well-known ports, e.g. sshd, pop3, squid, socks 5, etc
  - Allows us to leverage DNS without using SRV
- Networking in containers is point-to-point
  - No broadcast



Containers

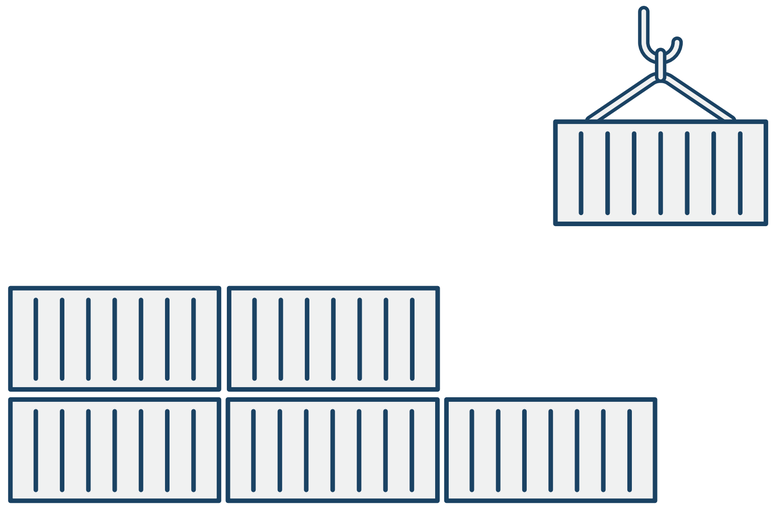
CTID	ADMST	OPST
112	UP	UP
113	UP	UP
124	UP	UP
132	UP	UP
176	UP	UP
192	UP	DOWN
129	UP	DOWN



Containers

Manager

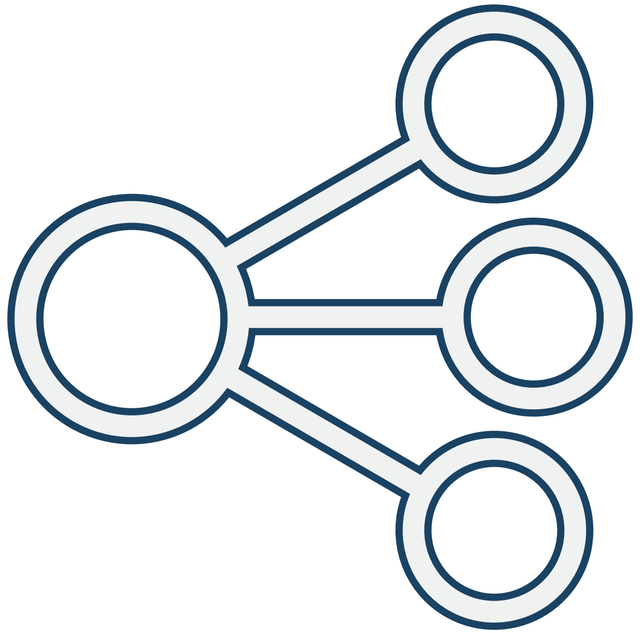
- Per Shard container manager
  - Agent on each HN broadcasting container state
  - Manager matches running state against configured state and takes appropriate actions
- Python based config framework & scheduler
  - OpenVZ configured via easy to create config files
  - Container config has resource allocations & mount properties
  - Compute node selected against high res monitoring.
  - Compute node can ‘reject’ a container for various reasons



Templates
Containers
DNS
Monitoring
Proxies
Cloud Peering

DNS

- Dynamic service discovery without App changes
- Where’s my mail server? It’s always at `‘smtp’`
  - POP3 server? Connect to “`mailstore`”
- Uses routing records and search path
  - CNAME to the real server
- Ops can move load around by changing DNS records



DNS

How To

- Create an internal facing DNS zone, e.g. `atlassian.net.local`
- Add service routing records in the database:

```
mailstore.<host>.atlassian.net.internal CNAME mailstore-101-1.rack-101.uc-inf.net
smtp.<host>.atlassian.net.internal      CNAME emx-101-1.rack-101.uc-inf.net
.. etc
```

- In container, add “`<hostname>.internal`” to the DNS search path
  - In OpenVZ, set ‘`SEARCHDOMAIN`’

DNS

How To

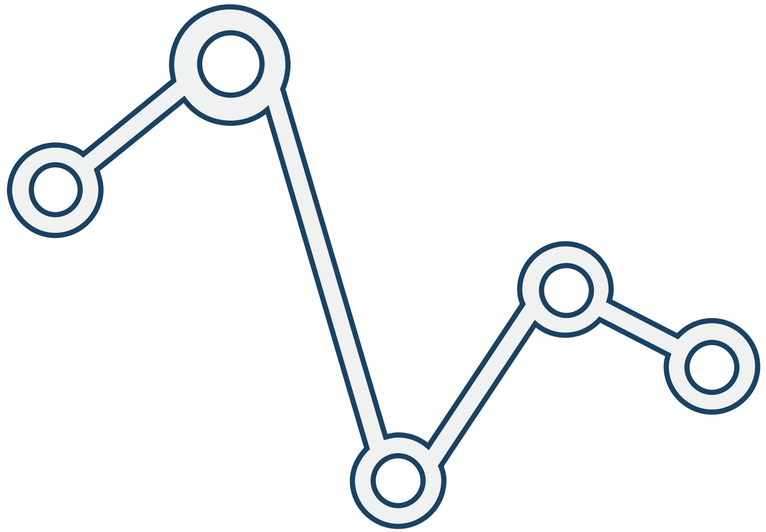
```
[dev][~]$ cat /etc/resolv.conf
search myhost.atlassian.net.internal
nameserver 10.10.10.10

[dev][~]$ host smtp
smtp.myhost.atlassian.net.internal is an alias for relay-101-1.sc1.uc-inf.net.internal
relay-101-1.sc1.uc-inf.net.internal has address 10.10.10.11
relay-101-1.sc1.uc-inf.net.internal has address 10.10.10.12
```

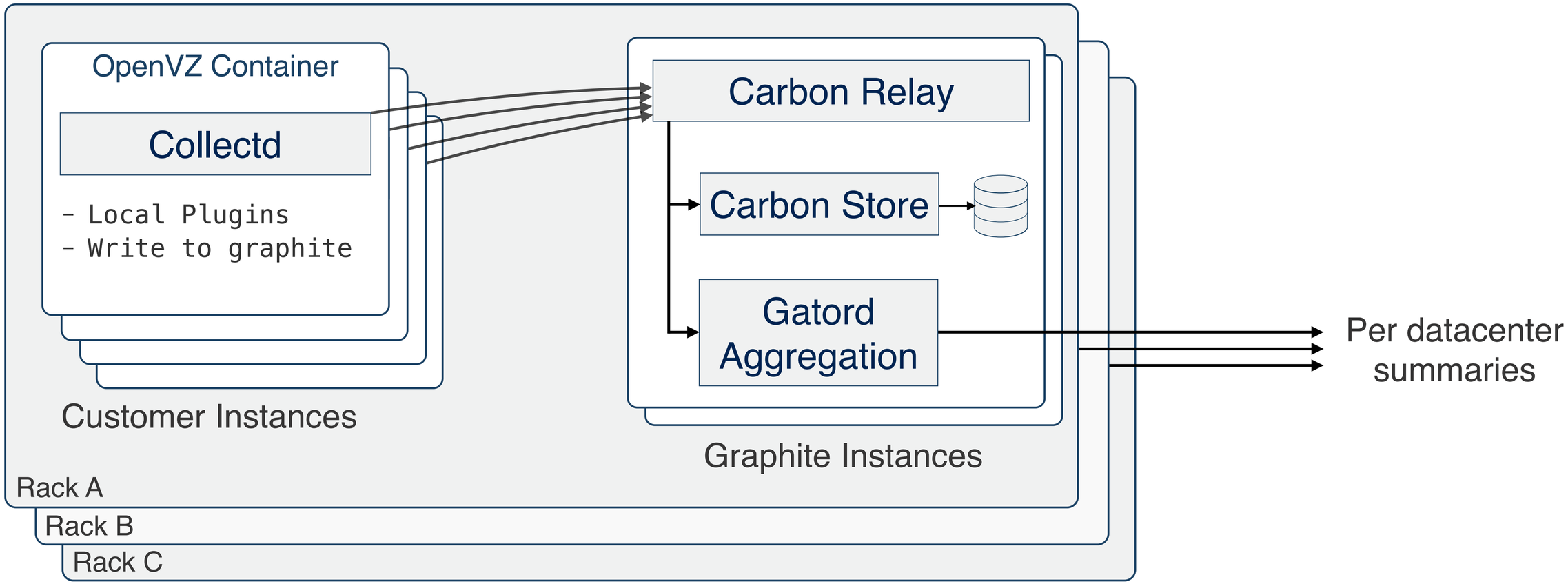
Templates
Containers
DNS
Monitoring
Proxies
Cloud Peering

Monitoring

- Instrument every process in the cluster
  - Gather CPU, memory, network, disk, etc
- Keep high resolution data and have fast summaries
- Dynamically configured as apps added/removed



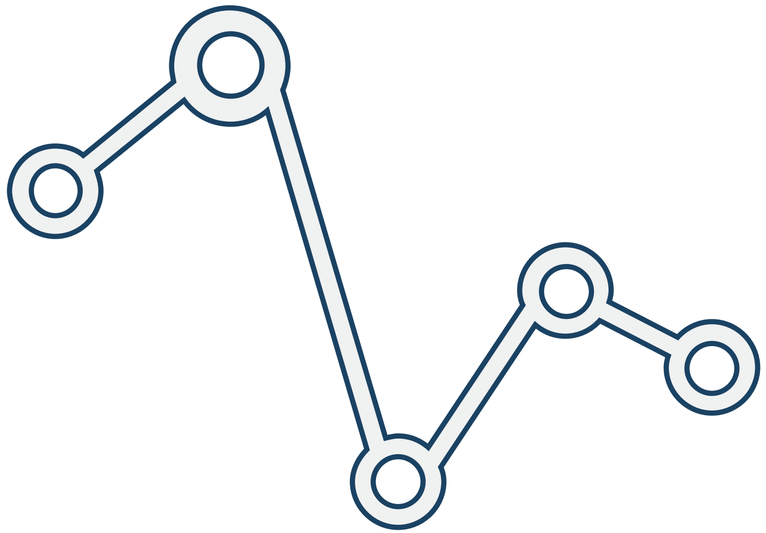
Monitoring


OpenVZ Container
Collectd
- Local Plugins
- Write to graphite
Customer Instances
Carbon Relay
Carbon Store
Gatord
Aggregation
Graphite Instances
Per datacenter
summaries
Rack A
Rack B
Rack C

Monitoring

How To - Client

- Run collectd in each container
  - Minimal config:
    - Graphite storage backend sending to “`metrics`”
    - Include a local config dir on /data
- Store a plug repo on /sw
  - Each plugin is tagged for a service: `#service_tag: httpd`
- Use inotify to watch /service for create, delete
  - Run a script which links the right plugins in



Monitoring

How To - Client

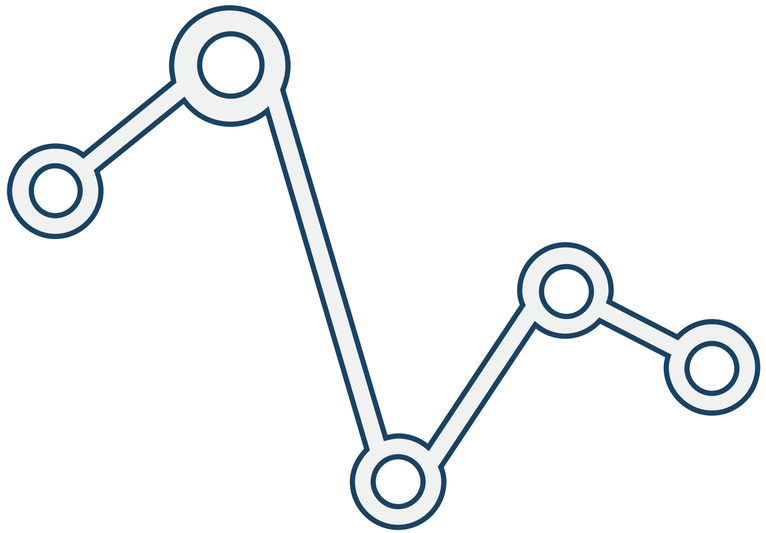
```
[prd][root@gbarnett:/data/sys_collectd/conf]# ls -al

postgres_db_confluence.conf
postgres_db_jira.conf
process_httpd.conf -> /sw/monitoring/collectd/conf/plugins/repo/process_httpd.conf
process_jvm_confluence.conf -> /sw/monitoring/collectd/conf/plugins/repo/process_jvm_confluence.conf
process_jvm_jira.conf -> /sw/monitoring/collectd/conf/plugins/repo/process_jvm_jira.conf
process_postgresql.conf -> /sw/monitoring/collectd/conf/plugins/repo/process_postgresql.conf
tail_apache.conf -> /sw/monitoring/collectd/conf/plugins/repo/tail_apache.conf
tail_jvm_confluence.conf -> /sw/monitoring/collectd/conf/plugins/repo/tail_jvm_confluence.conf
tail_jvm_jira.conf -> /sw/monitoring/collectd/conf/plugins/repo/tail_jvm_jira.conf
```

Monitoring

How To - Storage

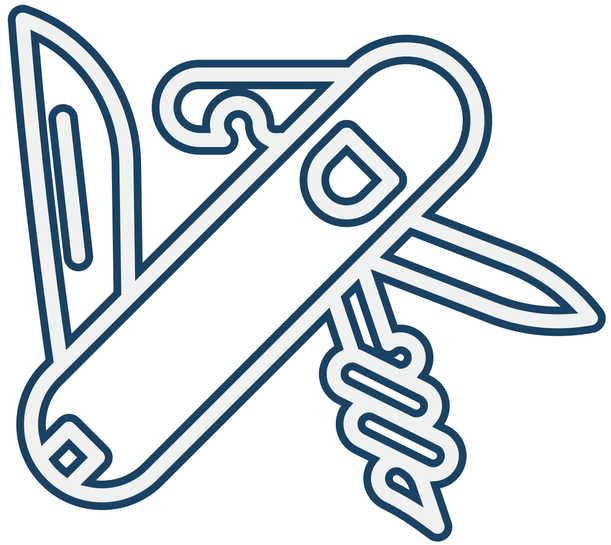
- Graphite / Carbon for metrics storage
- Multiple stores per rack, routing via DNS
- On each store, use carbon relay to split stream
  - 1 stream to disk for high resolution
  - 1 stream to summarisation daemon which forwards north
- Per DC top level carbon store for summary data



Templates
Containers
DNS
Monitoring
Proxies
Cloud Peering

Reverse Proxy

- Terminate inbound traffic on edge proxies
  - One place to enforce ACLs and HTTPs
  - Route URL paths to new backend services
  - Stateless configuration
- Multiple protocol support
  - Nginx for HTTP
  - Postfix for SMTP



Reverse Proxy

Nginx Example

- Default route to core container

```
location / {
        proxy_pass      http://$host.internal$request_uri;
}
```

- Experiments service to inject JS & CSS

```
location /experimental/ {
   rewrite ^/experimental/(.*)   /tenants/$host/jira/$1 break;
   proxy_pass http://experiments.$host.internal:8080$uri$is_args$args;
}
```

Reverse Proxy

Postfix Example

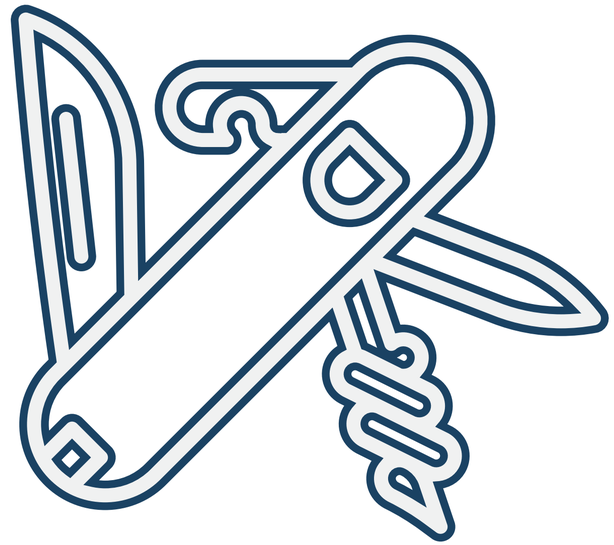
- Create mostly stateless email routers using tcp_lookup tables in config

```
transport_maps = tcp:127.0.0.1:2526
virtual_mailbox_maps = tcp:127.0.0.1:2527
virtual_mailbox_domains = tcp:127.0.0.1:2528
```

- Write a small daemon that answers lookups

```
get jira@myhost.atlassian.net
200 lmtp:mailstore.myhost.atlassian.net.internal:24

get doesntexist@myhost.atlassian.net
200 discard
```



Outbound Proxy

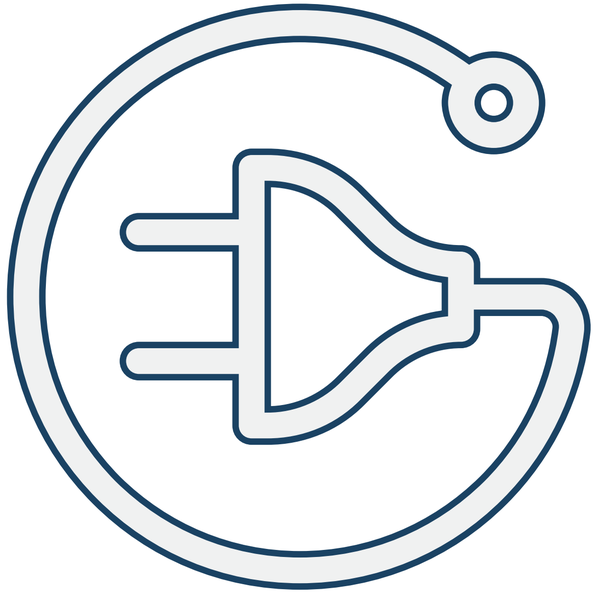
- Route all outbound connections via Socks 5 proxy
  - One place to apply ACLs and block thundering herds
  - ACLs per user (application) without needing source IP
  - Transparent using LD_PRELOAD and tsocks/dante
- Scale out
  - Request external connections from “`socks`”
  - Make more servers & change DNS routing records
  - Or scale behind a load balancer

Templates
Containers
DNS
Monitoring
Proxies

Cloud Peering

Cloud Peering

- Dedicated connection to cloud services
  - AWS Direct Connect, GCE Carrier Interconnect, etc
- High B/W, low latency access to APIs
  - General more stable and faster
- AWS VPCs can be presented as a Vlan tag
  - VPCs become like any other network segment
  - Possible on other providers using IPSEC VPNs



Templates
Containers
DNS
Monitoring
Proxies
Cloud Peering

Summary

- Pre-bake templates and fill later with capacity
- Use containers to wrap multi tenancy around the application.
- Remove static configs with DNS
- Support future decomposition using proxies and cloud peering

Thank you!

GEORGE BARNETT • SAAS PLATFORM ARCHITECT • ATLASSIAN

Image Attributions

The Noun Project:
CPU - Michael Anthony; Structure - Nikolay Necheuhin; Container - Creative Stall;
Pattern - Rohit Arun Rao; United States - Juan Pablo Bravo; Database - Anton Outkine;
API - Emily van den Heever; Graph - Tommy Lau; Socks - Milky Digital Innovation;
Knife - Nathan Driskell; Plug - Arthur Shlain