Stateful systems on immutable infrastructure

Hannu Valtonen

All Systems Go 2019 - Berlin

Speaker

@HannuValtonen

- VP Product, co-founder @ Aiven, a cloud company providing DBaaS and messaging services
- Previously: large scale database, distributed systems consultant, SW architect
- PostgreSQL pghoard, pgmemcache, pglookout maintainer

Google Cloud Platform

amazon web services

Microsoft Azure

UpCloud

DigitalOcean

packet

Background on Aiven

- *Your data cloud*
- Based in Helsinki, Boston, Sydney and now Berlin
- 8 data engines now available in 6 clouds and 87 regions, virtual and bare metal instances
- Launched a fully-managed PostgreSQL service in 2016
- Running nodes at scale globally

Some definitions

Stateful systems:

hold important data - the state

- that is typically heavy to move around
- needs to be available at all times
- must not get lost or altered accidentally

Contrast to stateless systems

- That are trivial to restart
- Trivial to move around
- Is fine if some are unavailable

Some definitions continued

Immutable infrastructure:

a paradigm in which servers are never modified after they’re deployed

- more consistent and reliable
- Simpler deployment process
  - Prevents configuration drift issues
- Requires quite a bit of tooling to pull off effectively

Base OS selection

- Team had a long history of using Debian
- We knew we didn’t want to do a lot of backporting
  - Especially of system components
- Preferably a distro that operates fairly close to upstream projects
- Good systemd integration desirable

Considerations around Debian

- Debian has a reputation for being "stable"
- Open Source/Free Software ethos taken _really_ seriously
    - No single controlling company (or anything close to it)
    - Lots of packages available (59k according to debian.org frontpage)
- Debian based Ubuntu very popular so lots of people have an idea of how to operate in the context

Debian not a perfect fit

- "Stable" in the Debian context means
    - ..releases were few and far between (especially earlier)
    - Need to either use packages from backports or to backport them yourself
    - Leading to having to backport package(s) system library dependencies
        - Which led to having a custom distro -> Why did we want "stable" again?
- While backporting packages in a limited quantity is fine, when you need to start doing it for system components (glibc, openssl, systemd)
    - you'd really rather be doing something else
- Debian at the time lacked good systemd integration

Enter Fedora

- 6 month release cadence
  - Sounded a bit scary at first
  - But at least all the packages are "fresh"
- Fairly fresh packages and libraries
  - ..but we still need things like fresh minor versions/patched versions built from git
  - We build ~150 packages on top of it
- systemd as a default for a while now
- .. and RPM spec files a joy to work with compared to .deb packaging

Out of the box Fedora experience

- An up-to-date kernel and systemd
- SystemD support done properly in packages (no init.d scripts, etc.)
- Mandatory Access Control via SELinux
- Firewall enabled and blocking everything
- No useless services, especially network listening ones, automatically enabled
    - Simply installing a package does not immediately configure and start serving at some port over public interfaces
- Python 3.7
    - Great starting point for building a cloud-based application

General Aiven philosophy on nodes

- Nodes (=VM or bare-metal machine) are disposable
  - They come and go at will
- Don't put any manual effort into any single node
  - We automate absolutely everything around a node’s lifecycle
- Don't rely on the underlying cloud providers infra too much
  - but take advantage of HW like local SSDs (we can because durability is handled otherwise)
- Handle durability by always persisting the data somewhere else (as well)
  - Generally either object storage (cheap), or other cluster nodes

Persistence and durability

- Since nodes can come and go at will, we don't want to rely on cloud providers EBS/Persistent Disk/Premium SSDs for persistence
  - because you can't move them that easily between clouds
  - We also get to enjoy local SSD performance where available
  - Which outperforms network SSDs quite a bit
    - EBS single volume max throughput 250 MB/s, upto 10k read IOPS
    - AWS i3 instance local SSDs north of 2GB/s, read IOPS between 100k/s and 3.3M/s depending on instance size

Persistence and durability continued

- Persistence for example handled in PostgreSQL with something like:
    - PostgreSQL backup daemon PGHoard storing WAL (Write Ahead Log) in semi-realtime to object storage. (creates a max boundary for data loss)
    - Replication, either asynchronous or synchronous to other cluster nodes diminishing the potential data loss to something between very little and nothing depending on the setup
        - Uses the service(s) own builtin replication instead of block level disk replication
- All nodes always separated to multiple AZs of a region
    - Data in clustered systems respects the AZ boundaries

Our approach to upgrades

- Rolling forward upgrades
  - Replace existing nodes either one by one or all at once by restoring/replicating data from backups/existing nodes
  - Once done, gracefully remove old nodes and do controlled failovers where applicable
- We do a lot of these every given day
  - Process has become fairly solid
- This is also what happens on SW and HW upgrades
- Same process works across cloud vendors/regions or when customer needs more HW capacity

Systemd-nspawn vs Docker

- Docker comes with its own set of baggage
  - systemd-nspawn is built in to the system already
  - systemd-nspawn has a more minimalistic, "unixy" approach to the problem
  - Our container images are basically just directory trees on disk
- nspawn unsurprisingly integrates well with systemd
  - systemd unit files (We use plenty of unit directives)
  - Journald (structured logging)

Host machine

- Host machine (VM or bare-metal) is running Fedora too
    - Runs a single container for customer services
- On management agent startup it refreshes the pre-installed packages on images
    - Though we build our images often this allows us to force immediate updates of any single package on all new machines
    - Typically there’s nothing to do with this step since everything is up to date already
- After this point the host system retains the same software versions for the duration of its lifetime and is immutable

Management agent (pruned)

- Once started up builds the customer service according to configuration given
  - Disk layout setup (RAID, encryption)
  - Sets up a service cluster internal IPSec IPv6 network to other nodes
  - Setups service itself with a correct configuration including user overrides
  - Restores data from backups/other nodes
  - Monitors and reports service health upstream
  - Reacts to configuration changes (changed nodes, changed configuration)
  - Sets up auxiliary management agents

Management agent (pruned) continued

- Agents run on host-side
  - our management agent which handles all configuration changes and management
  - metrics agent (telegraf https://github.com/influxdata/telegraf)
  - Log shipping agent (journalpump, https://github.com/aiven/journalpump)
  - backup/HA daemons (mostly Open Source(d) and written by us)
- All our communications are over Apache Kafka

Container for user services

- Runs within a fairly locked down systemd-nspawn
- Contains only customer end-user services
    - PostgreSQL, Apache Kafka, etc.
    - None of which allow code execution by default
- After installation, container is totally immutable except for:
    - config files for the services run there
    - data files on data volumes

Image building

- Support for six different cloud providers makes this a bit tricky
  - All clouds have different ways to create and register new images
  - Some like DigitalOcean don't allow you to actually upload your own but to only customize their base images + snapshot
- Our build automation needed to adopt multiple strategies for building the image
- In some public clouds this takes much longer (Azure)
  - Need to transfer the image to all the regions of that cloud vendor
- We now have 89 cloud regions supported in total at the time of writing

Pre-installed packages on images

- Many of our software packages are large (X hundred meg RPMs)
- Any kind of installation activity at node boot delays getting the node to serve the customer
  - Prefer having all packages pre-installed on the cloud boot image
- Because of this our startup is fairly fast
  - Depending on cloud provider ~2-10 minutes from node creation to being able to serve the customer

Testing

- Updating system base software this much and this often can break things
  - Need to have a large test suite
    - We have caught many issues over time by some test x starting to fail
- A bit of continuous pain involved with having a fast moving underlying distro
  - Still preferred to the "immense amount of pain every couple of years"-approach
- Read the release notes of everything with a magnifying glass

Not always smooth sailing

- Got especially badly bitten by the semi-recent glibc unicode changes
  - We were aware of the collation changes but were expecting them in the next glibc version, didn't realize it had been backported to the previous release in Fedora
- IPSec keeps on breaking in different ways
- random software breaks at times
  - Including systemd-nspawn (machinectl shell used to be broken in Fedora for a while)
- DNF is a bit on the slow side
  - ...and not resilient against temporary network errors - we use a wrapper

But in general...

- Very happy with Fedora
  - allowed us to focus on our services instead of system component maintenance
- Also forced us to de-emphasize the meaning of any single node
- VMs and containers are totally disposable for us
  - The value we provide doesn’t come from hand massaging an existing node
  - Need to take care of persistence otherwise
- Automation of pretty much everything a must for a model like this

Questions?

Any questions?

(First x questions get a pair of Aiven socks)

..and we're hiring for our brand new Berlin office

aiven

Thanks!

🌐 https://aiven.io

🐦 @aiven_io

🐦 @HannuValtonen