Resilient puppet in the cloud

OSAD 2022

Dr. Jan Bundesmann

October 4th, 2022

Agenda

1. Cloud & Puppet: Is this a thing?
2. Stacked puppetserver infrastructure
3. Target focussing: Agents as stacks & units
4. Conclusions and Outlook

Agenda

1. Cloud & Puppet: Is this a thing?
2. Stacked puppetserver infrastructure
3. Target focussing: Agents as stacks & units
4. Conclusions and Outlook

Why wouldn’t it be a thing?

- Old-fashioned: no cloud when Puppet was invented
- Slow: 30min heartbeat vs. seconds in GitOps

Expectation management

- No immediate spin-up
- No immutable infrastructure

- auto-deployment
- arbitrarily complex infrastructure

- No pets, no cattle
- Manageable complexity

Agenda

1 Cloud & Puppet: Is this a thing?

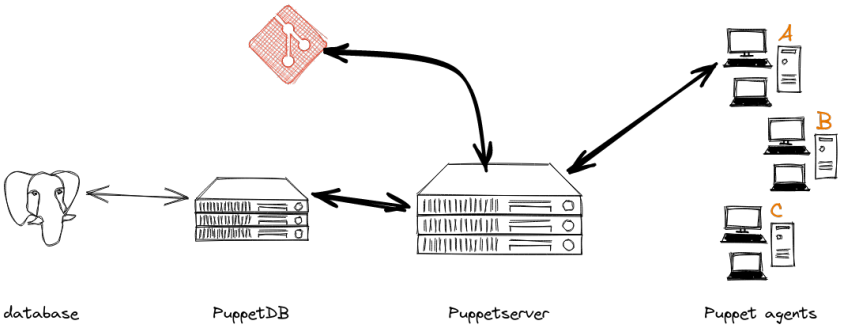
2 Stacked puppetserver infrastructure

3 Target focussing: Agents as stacks & units

4 Conclusions and Outlook

What? Stacks?

Stacks: components of (distributed) applications, consisting of one or more units



Why? Stacks?

- De-individualize instances
  - Care for working units
  - Redeploy non-working units
  - Stack broken?
- For puppet: use cloud mechanisms, map cloud features
  - DBaaS, GITaaS
  - Scalability
  - Identity and Access Management, Secret Management

How? Stacks?

- Single source of thruth
- One CA, multiple puppetservers
  - Access rights
  - Backup
  - Unique identifiers
- Race conditions
  - Tolerance for DB inaccessibility
  - Logging / Monitoring: expected failures (regarding PuppetDB in particular)
- Puppet manages everything

How? Stacks?

- Single source of thruth → control repository, r10k
- One CA, multiple puppetservers
  - Access rights
  - Backup
  - Unique identifiers
- Race conditions
  - Tolerance for DB inaccessibility
  - Logging / Monitoring: expected failures (regarding PuppetDB in particular)
- Puppet manages everything

How? Stacks?

- Single source of thruth → control repository, r10k
- One CA, multiple puppetservers → shared folder, each instance considers itself singular
  - Access rights
  - Backup
  - Unique identifiers
- Race conditions
  - Tolerance for DB inaccessibility
  - Logging / Monitoring: expected failures (regarding PuppetDB in particular)
- Puppet manages everything

How? Stacks?

- Single source of thruth → control repository, r10k
- One CA, multiple puppetservers → shared folder, each instance considers itself singular
  - Access rights
  - Backup
  - Unique identifiers
- Race conditions → Eventual consistency through foreman modules
  - Tolerance for DB inaccessibility
  - Logging / Monitoring: expected failures (regarding PuppetDB in particular)
- Puppet manages everything

How? Stacks?

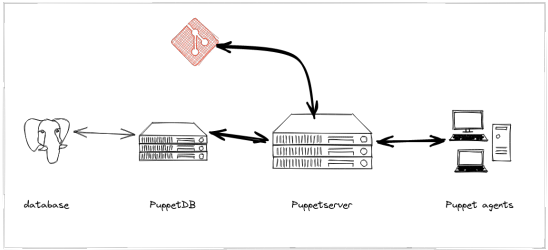
- Single source of thruth → control repository, r10k
- One CA, multiple puppetservers → shared folder, each instance considers itself singular
  - Access rights
  - Backup
  - Unique identifiers
- Race conditions → Eventual consistency through foreman modules
  - Tolerance for DB inaccessibility → `soft_write_failure`
  - Logging / Monitoring: expected failures (regarding PuppetDB in particular)
- Puppet manages everything

How? Stacks?

- Single source of thruth → control repository, r10k
- One CA, multiple puppetservers → shared folder, each instance considers itself singular
  - Access rights
  - Backup
  - Unique identifiers
- Race conditions → Eventual consistency through foreman modules
  - Tolerance for DB inaccessibility → `soft_write_failure`
  - Logging / Monitoring: expected failures (regarding PuppetDB in particular)
- Puppet manages everything → next section

Buzzwords and stats

- Infrastructure is
  - Ephemeral
  - Idempotent
  - Not Immutable
- Cloud manages scaling of stacks
  - Autoscaling, load balancing
  - Unique identifiers
- Redeployed within 20 minutes
  - Raw OS images
  - Accelerate with your specialized images



Agenda

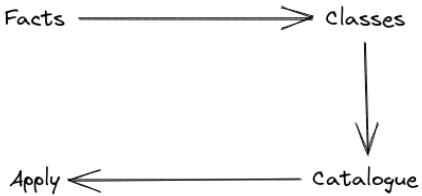
1 Cloud & Puppet: Is this a thing?

2 Stacked puppetserver infrastructure

3 Target focussing: Agents as stacks & units

4 Conclusions and Outlook

How does puppet work



How to choose classes

- Server chooses (very old)

```
node 'www1.example.com' {
  include common
  include apache
  include squid
}
```

- Server chooses (nowadays, using hiera lookup)

```
hiera_include('classes')
```

Determine classes from hiera

hiera.yaml:

```yaml
---
version: 5
defaults:
hierarchy:
  - name: "Per-node data"
    path: "nodes/%{::trusted.certname}.yaml"
  - name: "Other YAML hierarchy levels"
    path: "common.yaml"
```

Hiera structure:

```
.
+-- common.yaml
+-- nodes
    +-- www1.example.com.yaml
    +-- www2.example.com.yaml
```

common.yaml:

```yaml
classes:
  - common
```

nodes/www1.example.com.yaml:

```yaml
classes:
  - apache
  - squid
```

nodes/www2.example.com.yaml:

```yaml
classes:
  - nginx
  - certbot
```

Focus on your facts

- Write facts
  - custom facts (written in ruby)
  - external facts (written in any language)
  - hostname based
  - ip-range based
  - query API of inventory system
  - simple plain JSON or YAML files
- Which facts determines the stack?
- Use hiera to aggregate classes as in roles, profiles or **stacks**!

Obtain list of profiles using `mapped_paths`

```yaml
---
version: 5
defaults:
hierarchy:
  - name: "roles"
    mapped_paths:
      - stacks
      - stack
      - "stack/%{role}.yaml"
  - name: "Generic defaults"
    path: "common.yaml"
```

```
.
+-- common.yaml
+-- stack
    +-- jitsi.yaml
    +-- pizza_delivery.yaml
    +-- puppetdb.yaml
    +-- puppetserver.yaml
    +-- wordpress.yaml
```

Example: puppetserver

/etc/facter/facts.d/roles.json:

```
{
  "stacks": [
    - "puppetserver"
  ]
}
```

puppetserver.yaml

```
---
classes:
  - puppet
  - puppet::server::puppetdb

puppet::server: true
puppet::server_ca: true

# important: the module by default tries to contact a foreman instance
# the following three parameters prevent this
puppet::server_foreman: false
puppet::server_foreman_facts: false
puppet::server_external_nodes: ''

# settings for storing catalogs and facts in puppetdb
puppet::server_storeconfigs: true
puppet::server_reports: store,puppetdb
puppet::server::puppetdb::server: puppet-db.server.in.the.cloud
puppet::server::puppetdb::soft_write_failure: true
```

Example: Application

/etc/facter/facts.d/roles.json:

```
{
  "stacks":  "wordpress"
}
```

wordpress.yaml

```
---
classes:
  - wordpress
  - reverse_proxy

reverse_proxy::virtual_server:
  - name: %{::facts.fqdn}
    proxy_backend: http://localhost:8080

wordpress::installed_plugins: recipes
```

Agenda

1 Cloud & Puppet: Is this a thing?

2 Stacked puppetserver infrastructure

3 Target focussing: Agents as stacks & units

4 Conclusions and Outlook

Summary

- De-cluster using stacks!
- Use stacks to strengthen and fasten your infrastructure!
- Empower your agents to choose stacks!