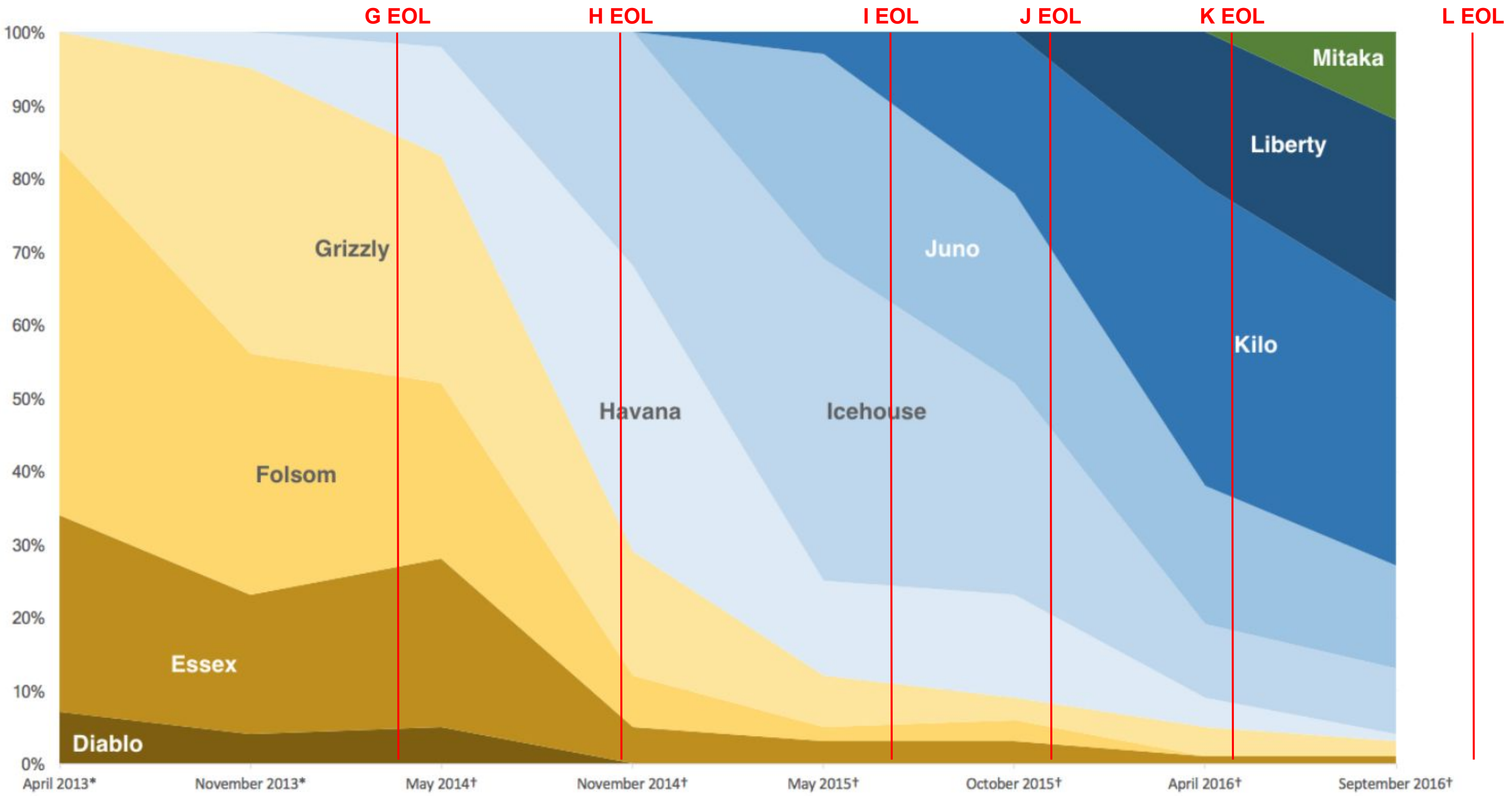
Upgrading OpenStack

Michał Dulko
Artur Korzeniewski

OpenStack upgrade trends



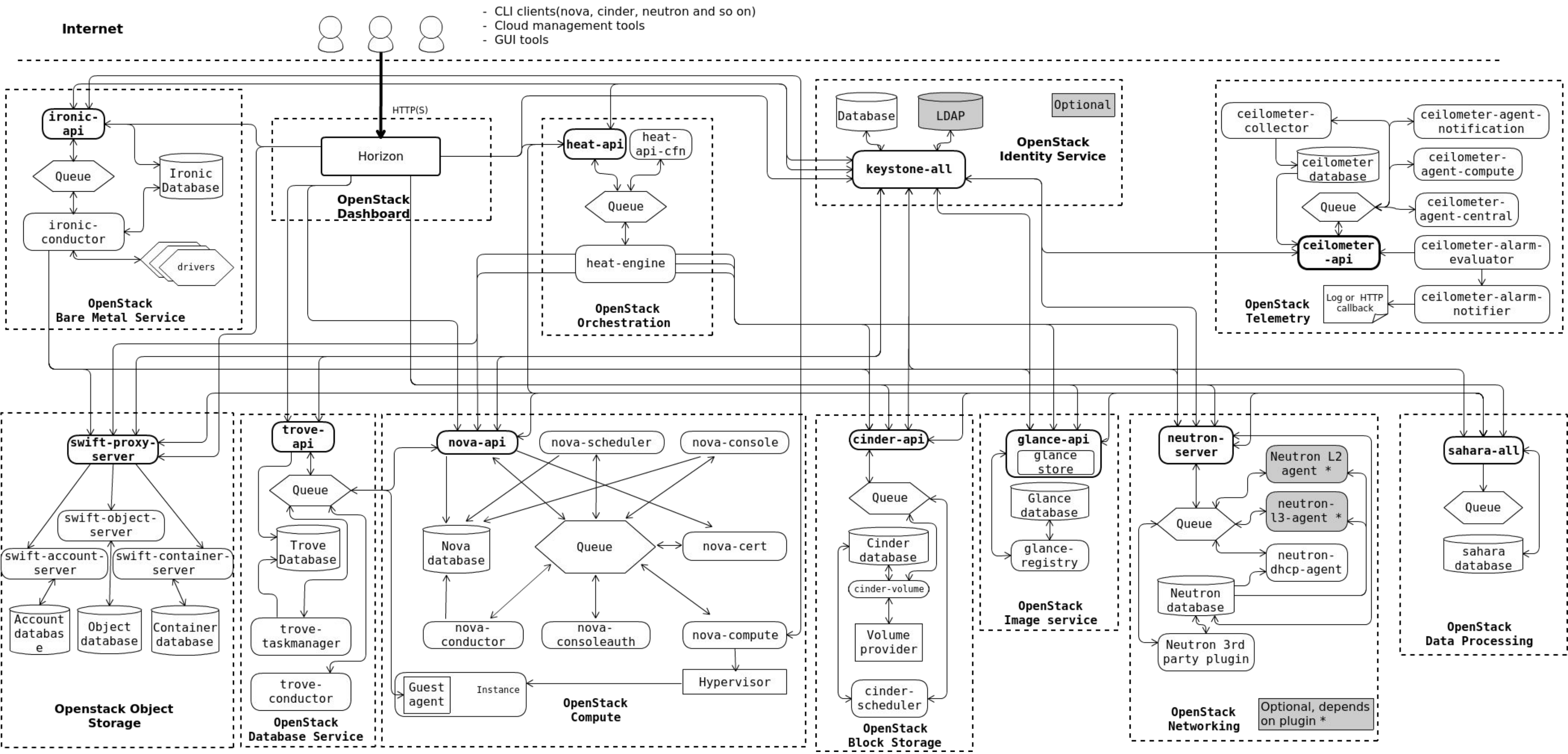
"*OpenStack User Survey - Deployments-only update (October 2016)*" by OpenStack Foundation is licensed under CC BY-ND 4.0; Source: https://www.openstack.org/assets/survey/October2016SurveyReport.pdf

OpenStack release and support trends

- OpenStack will most likely continue to be released in 6-month cycles.
  - Community discussion always ended in favor of keeping current release schedule.
- The release support period will probably continue to be 1 year.
  - Main issue - keeping CI/CD functional on older releases.
  - LTS releases are currently out of question.
  - Extended support is provided by vendors.

Rolling upgrades of OpenStack

It’s all about compatibility



Internet

- CLI clients(nova, cinder, neutron and so on)
- Cloud management tools
- GUI tools

HTTP(S)

ironic-api

Queue

Ironic Database

ironic-conductor

drivers

OpenStack Bare Metal Service

Horizon

OpenStack Dashboard

heat-api

heat-api-cfn

Queue

heat-engine

OpenStack Orchestration

Database

LDAP

Optional

keystone-all

OpenStack Identity Service

ceilometer-collector

ceilometer database

Queue

ceilometer-api

ceilometer-agent-notification

ceilometer-agent-compute

ceilometer-agent-central

ceilometer-alarm-evaluator

ceilometer-alarm-notifier

Log or HTTP callback

OpenStack Telemetry

swift-proxy-server

swift-object-server

swift-account-server

swift-container-server

Account database

Object database

Container database

Openstack Object Storage

trove-api

Queue

Trove Database

trove-taskmanager

trove-conductor

OpenStack Database Service

nova-api

nova-scheduler

nova-console

Nova database

Queue

nova-cert

nova-conductor

nova-consoleauth

nova-compute

Hypervisor

Guest agent

Instance

OpenStack Compute

cinder-api

Queue

Cinder database

cinder-volume

Volume provider

cinder-scheduler

OpenStack Block Storage

glance-api

glance store

Glance database

glance-registry

OpenStack Image service

neutron-server

Neutron L2 agent *

neutron-l3-agent *

Queue

neutron-dhcp-agent

Neutron database

Neutron 3rd party plugin

OpenStack Networking

Optional, depends on plugin *

sahara-all

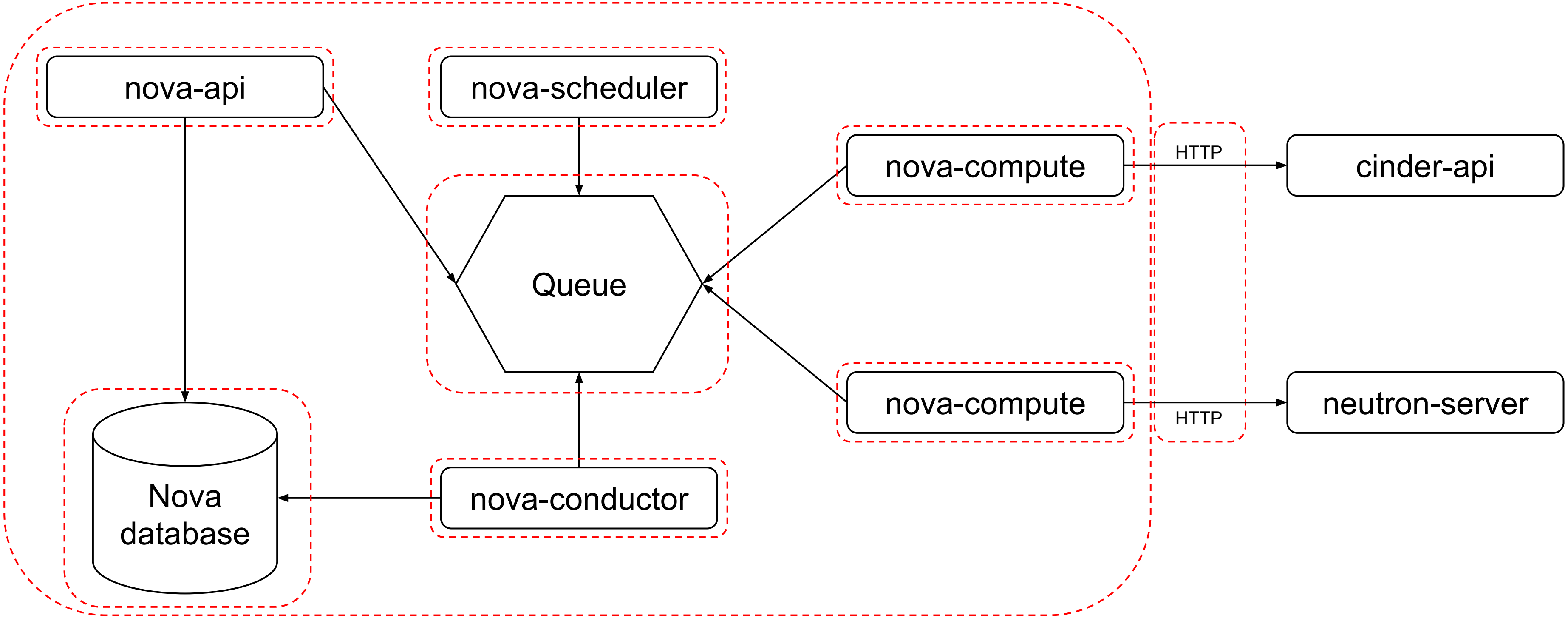
Queue

sahara database

OpenStack Data Processing

Diagram by OpenStack Foundation is licensed under Creative Commons Attribution 3.0 License; Source: http://docs.openstack.org/admin-guide/common/get-started-logical-architecture.html

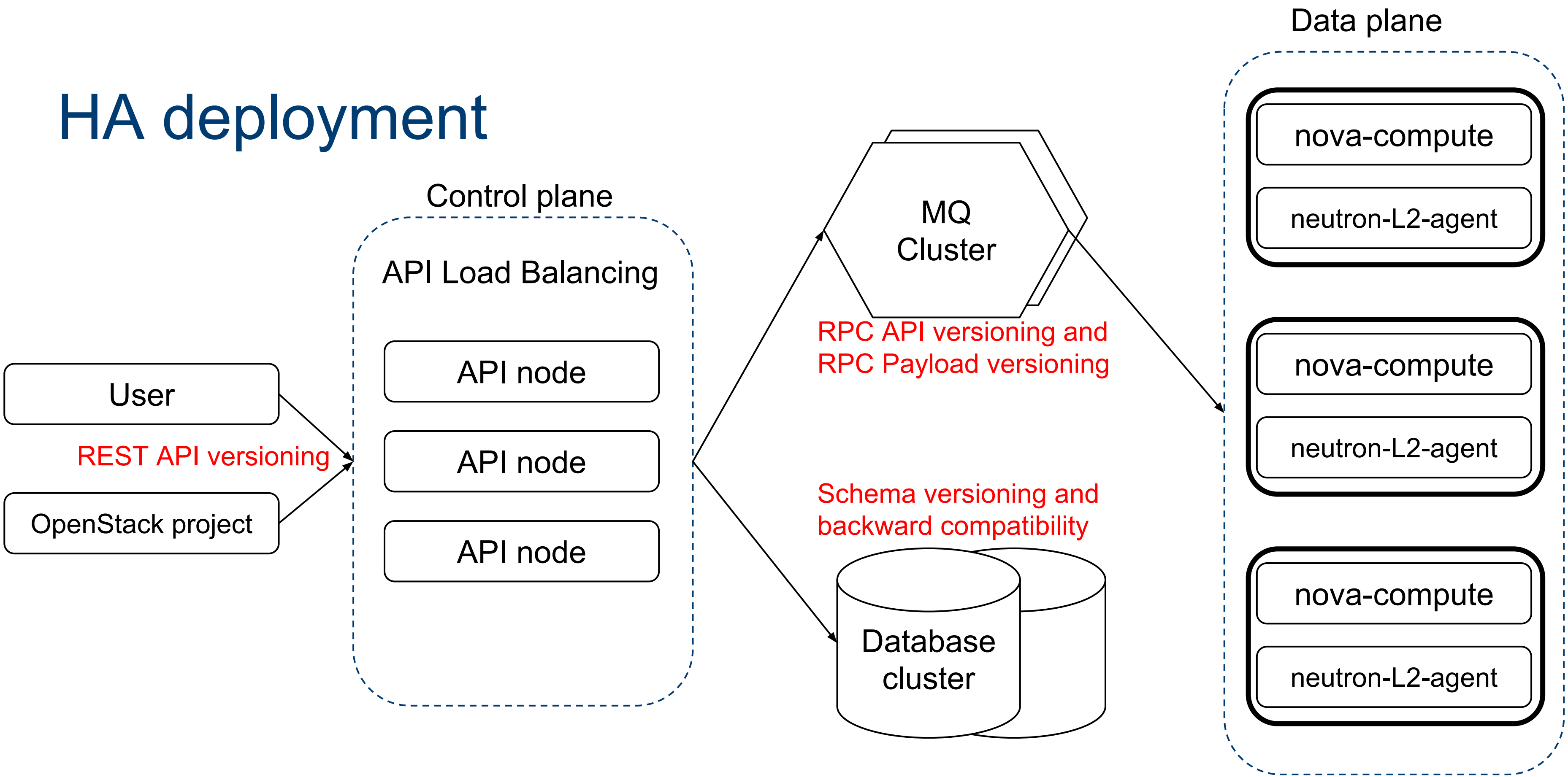
OpenStack services overview



Assumptions

- No release skipping.
  - From release X you can live upgrade to X+1, but not to X+2.
- Continuous deployment from master is allowed.
  - Some restrictions can apply.
- Resolving dependency hell is left to operator.
  - Use Docker containers, Python venvs, anything.
- You need HA deployment to minimize downtime during upgrade.
  - Non-HA deployments will certainly experience control plane downtime.

HA deployment



REST API versioning

RPC API versioning and
RPC Payload versioning

Schema versioning and
backward compatibility

Compatibility layers

- REST API
- RPC API
- RPC payload
- Database schema

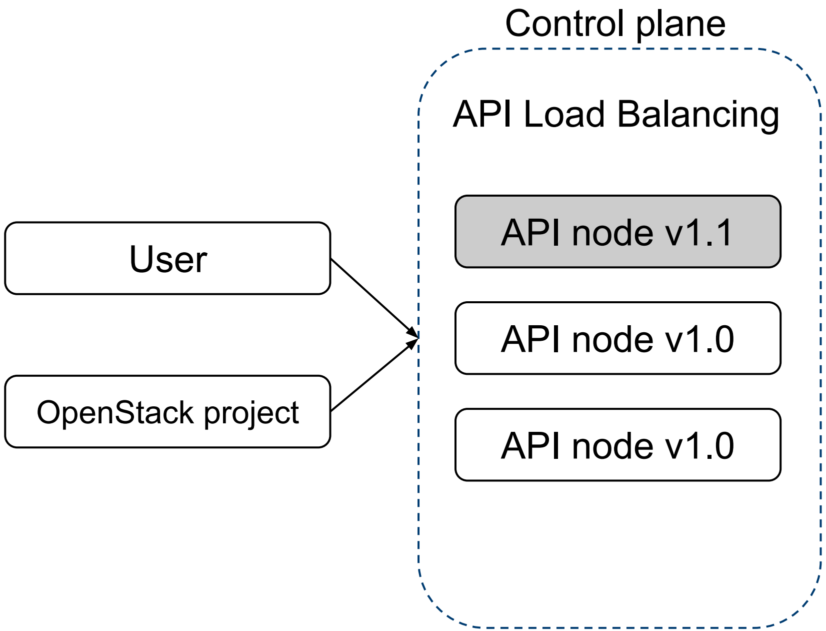
REST API

- User-facing REST API should be always compatible until depreciation.
- Any modification of REST API should require version bumping
- API extensions in Neutron
  - Adding new field for extension does not change API version
  - API for querying which extensions is available: http://neutron-api/v2.0/extensions
  - Introduce vendor-specific functionality to common networking resources

REST API - microversioning

- Some of the projects (Nova, Cinder, Ironic) adopted microversions.
- Any API change is versioned and client needs to explicitly state which microversion he expect.
- Server will keep support for microversions from at least 2 releases.
  - In practice much longer.
- There is API for microversion discovery, so clients can detect which APIs are available.

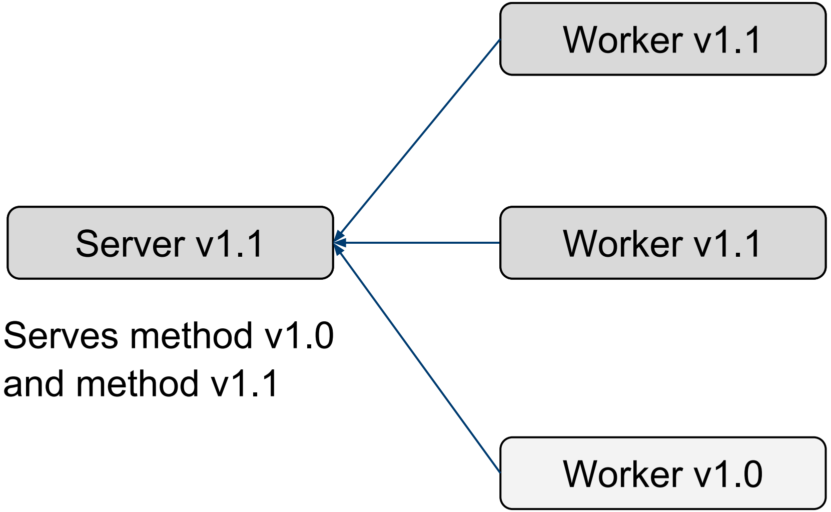
REST API - Load balancing



- Restart all API endpoints at once
- Exclude new version of API till all nodes are upgraded
- Pin version of API to lowest available

RPC API

- Changes in RPC API are versioned - method signatures.
- Servers are reporting their supported RPC API versions.
- Clients can then negotiate which RPC API version to use.
- Then internal communication between microservices can keep compatibility.

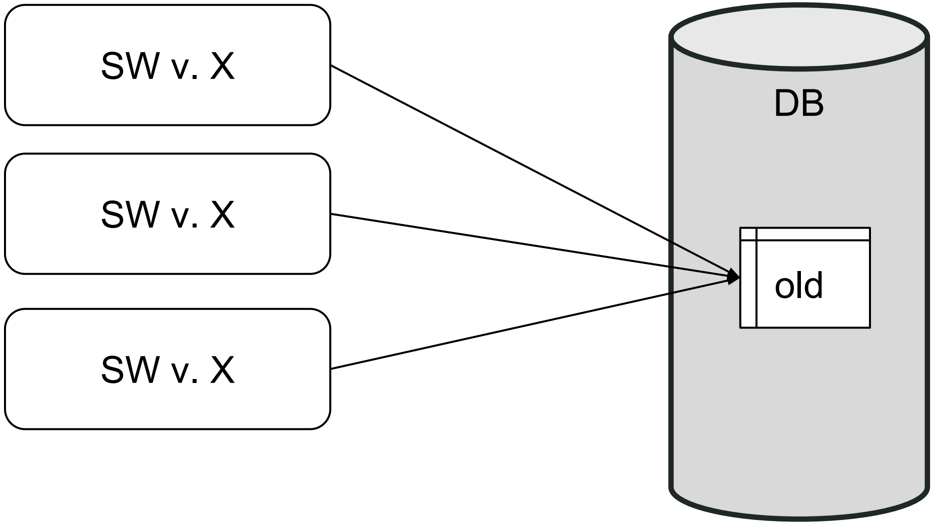


Serves method v1.0 and method v1.1

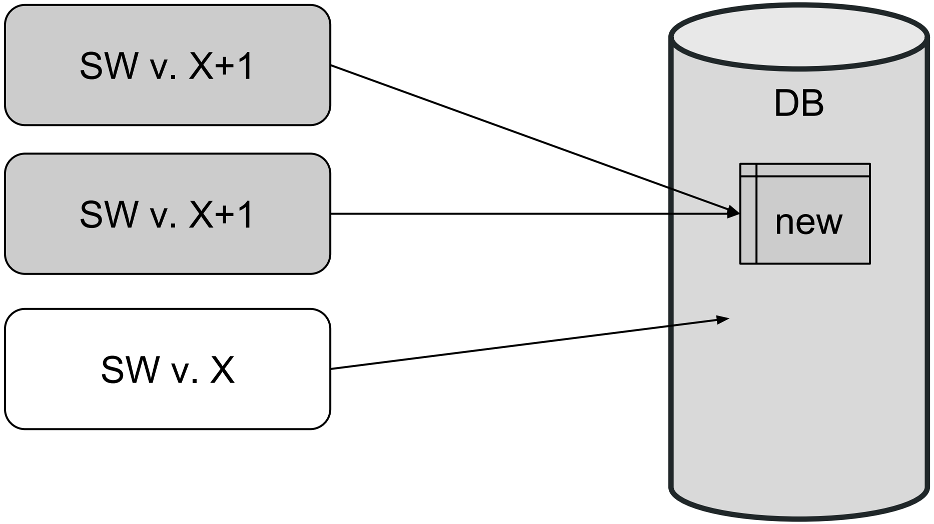
RPC payload

- Complex payloads (objects and dictionaries) are versioned.
    - oslo.versionedobjects library (OVO)
- Backward compatibility of objects:
    - Sent to be backported on receiving (Nova).
        - This means more RPC traffic during the upgrade.
    - Backported before sending to lowest common denominator (Cinder).
    - Sent in all required/registered versions (Neutron).

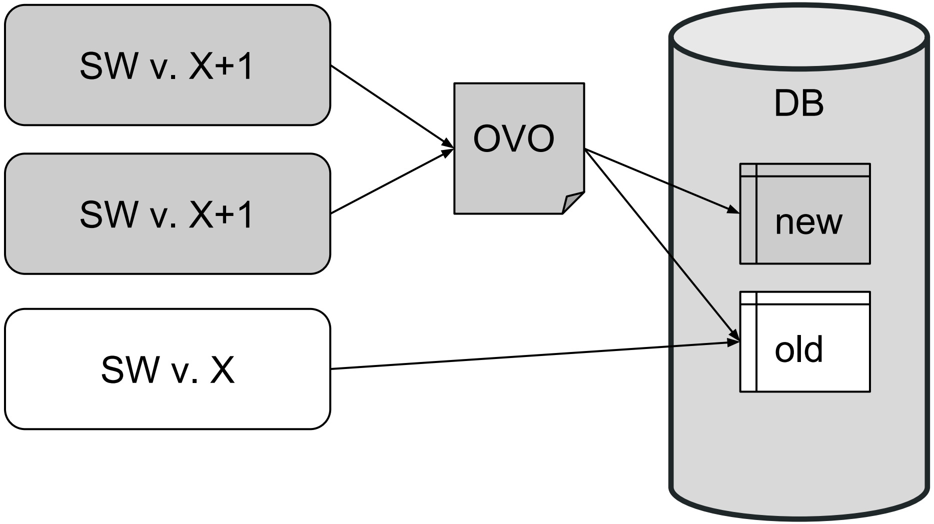
Database schema



Database schema - backward compatibility



Database schema - backward compatibility



Database schema

- Only backward-compatible changes allowed.
  - Instead of removing/renaming a column - create a new one and move data.
- No data migrations in schema migrations. Either:
  - Data to be migrated online when upgrade is finished (Nova, Cinder, Neutron).
  - Data migrated/synchronized using DB triggers (Keystone, Glance).
- No changes that involve table rewriting or locking.
  - E.g. resizing `VARCHAR` from 255 to 256 in MySQL.

Cloud upgrade procedure

General considerations

- Per-project approach.
  - Although some procedures are shared, every project has it’s own upgrade steps.
  - Project-specific instructions can be found in OpenStack Operations Guide.
    - https://docs.openstack.org/ops-guide/ops-upgrades.html#service-specific-upgrade-instructions
  - Also read *“Upgrade Notes”* section in each project’s release notes.
    - https://releases.openstack.org/ocata/index.html
- CERN publishes their upgrades insights from Havana to Liberty.
  - OpenStack in production blog
    - http://openstack-in-production.blogspot.com/

Rolling upgrade status in projects

	Nova	Swift	Neutron	Cinder	Keystone	Glance	Ironic
I→J	🟩	🟩	🟥	🟥	🟥	🟥	🟥
J→K	🟩	🟩	🟥	🟥	🟥	🟥	🟥
K→L	🟩	🟩	🟩	🟥	🟥	🟥	🟥
L→M	🟩	🟩	🟩	🟩	🟥	🟥	🟥
M→N	🟩	🟩	🟩	🟩	🟥	🟥	🟥
N→O	🟩	🟩	🟩	🟩	🟩	🟨 Experimental	🟥

Project upgrade order

Operations Guide order:

1. Keystone
2. *Swift*
3. Glance
4. Nova
5. Neutron
6. Cinder
7. Horizon
8. *Ironic*

Source: Upgrades chapter in OpenStack Operations Guide

OpenStack-Ansible order:

1. Keystone
2. Glance
3. Cinder
4. Nova
5. Neutron
6. Horizon
7. Swift
8. Ironic

Source: setup-openstack.yaml playbook in OSA repo

Nova’s procedure - Before maintenance window

1. Backup everything
2. Try DB migrations (both schema and data) on an offline copy of your DB
   - `nova-manage db sync && nova-manage api_db sync`
   - `nova-manage db online_data_migrations`
3. Run schema migrations on your production DB
   - `nova-manage db sync && nova-manage api_db sync`

Nova’s procedure - During maintenance window

1. Stop control plane services
   - Order: nova-api, nova-conductor, nova-scheduler
2. Upgrade them with new code and dependencies
   - If you’re using containers - you’re just swapping them
3. Start them again in reversed order
   - `[upgrade_levels]compute=auto`
4. Similarly upgrade nova-compute instances one-by-one
   - `[upgrade_levels]compute=auto`

Nova’s procedure - After maintenance window

1. Check if there are no orphaned services
   - `nova service-list`
2. Send SIGHUP signal to each of the services
   - This invalidates services version cache
3. Review your configuration for deprecated options
4. Run online data migrations
   - `nova-manage db online_data_migrations --limit <number>`

Automated tools - benefits

- Dependency isolation
  - Upgrading one service does not break others on the same node
- Upgrade orchestration
  - Easy to manage the upgrade procedure by scripts and build-in functionality
- Kubernetes based solutions advantages:
  - Self-healing
  - Easy scaling
  - Load balancing and HA of deployment

Automated tools

- Kolla
  - Kolla-ansible
    - First approach
    - Installing docker containers using Ansible
  - Kolla-kubernetes
    - New functionalities in Kubernetes enabled Kolla to start integration
    - Take advantage of Kubernetes microservices management lifecycle

Automated tools

- Kubernetes based installers
  - OpenStack Helm
    - Helm charts to install OpenStack
    - Image agnostic solution
  - Fuel CCP - Containerized Control Plane
    - Python based framework
    - Image building and bootstrapping

Automated tools

- OpenStack Ansible
  - Using LXC containers to isolate packages
  - Ansible scripts deploying LXC containers
  - Orchestrated upgrades by Ansible playbooks

Summary

To execute a successful rolling upgrade of OpenStack cloud from X to X+1:

1. Read projects documentation, release notes (both X and X+1), operators blogs and openstack-operators mailing list.
2. Try DB migrations on offline copies of your DB's.
3. Backup everything before starting.
4. If you're using some deployer software, check if it supports upgrading.
5. **Report any bugs found to Launchpad.**

Q&A

Slides available on GitHub:

https://github.com/dulek/openstack-day-poland-upgrades

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.