Service Assembly and Composition in CORD – An XOS Tutorial –

CORD
Central Office Re-architected as a Datacenter

#OpenCORD

Tutorial Outline

Architectural Overview

- Internal Structure / Interfaces / Control Flow / Diagnostic Tools

Data Model

- Core Models / Extending Models / Modeling Services

Assembling a Service

- Defining Models / Synchronizer / Extending APIs

Configuration Management

- Elements of a Config / Existing Configurations / Demo

Sources of Information (1)

XOS Developer Guide

- http://guide.xosproject.org/devguide/
- *ExampleService Tutorial*
- *Data Modeling Conventions*

CORD Design Notes

- http://opencord.org/collateral
- *Service Assembly and Composition in CORD*
- *CORD Reference Implementation*

Sources of Information (2)

Source Code

https://github.com/open-cloud/xos

**/README.md*

Also…

Email: https://groups.google.com/a/xosproject.org/forum/#!forum/devel

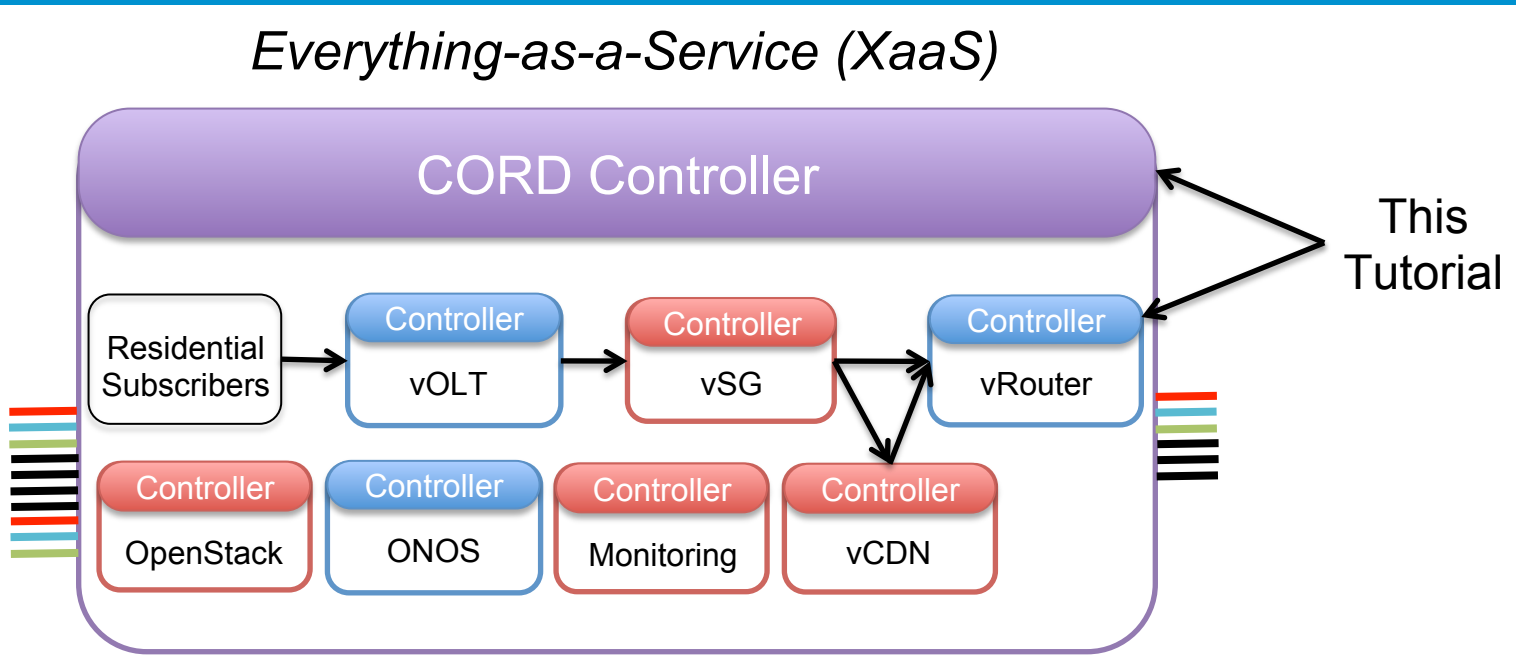
Slack: https://slackin.opencord.org/

REST API: http://docs.xos.apiary.io/

Lifecycle Management

- Design Time Framework
  - Organized as a graph of Multi-Tenant Cloud Services
  - On-boarding Services is a Distinct Step
    - Service Developer Provides Service Elements
    - Operator Configures Service Graph
- Runtime Execution Framework
  - Interface to Provision and Scale Services
  - Interface to Instantiate and Control Service Instances
  - Interface to Monitor Service Performance & Behavior

CORD Software Architecture



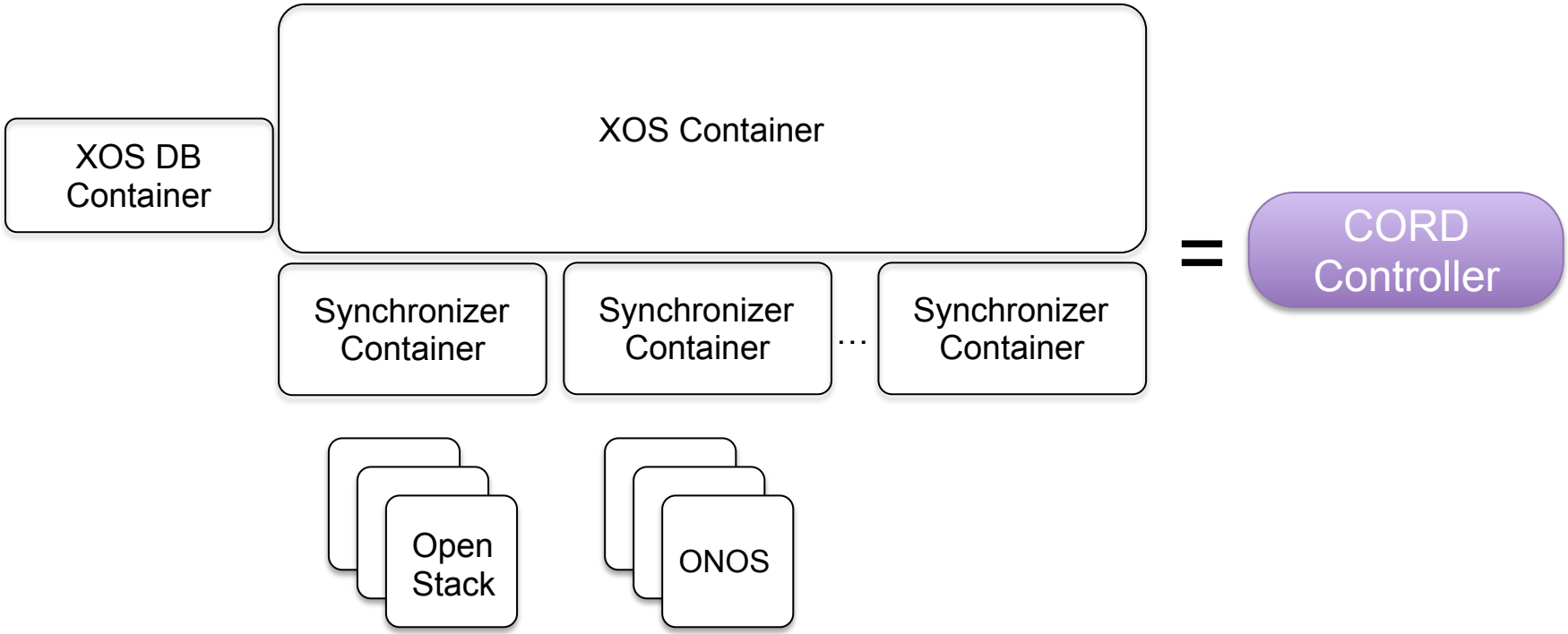
Key Takeaways…

(1) Service Controllers consolidate (and make explicit) all configuration and control

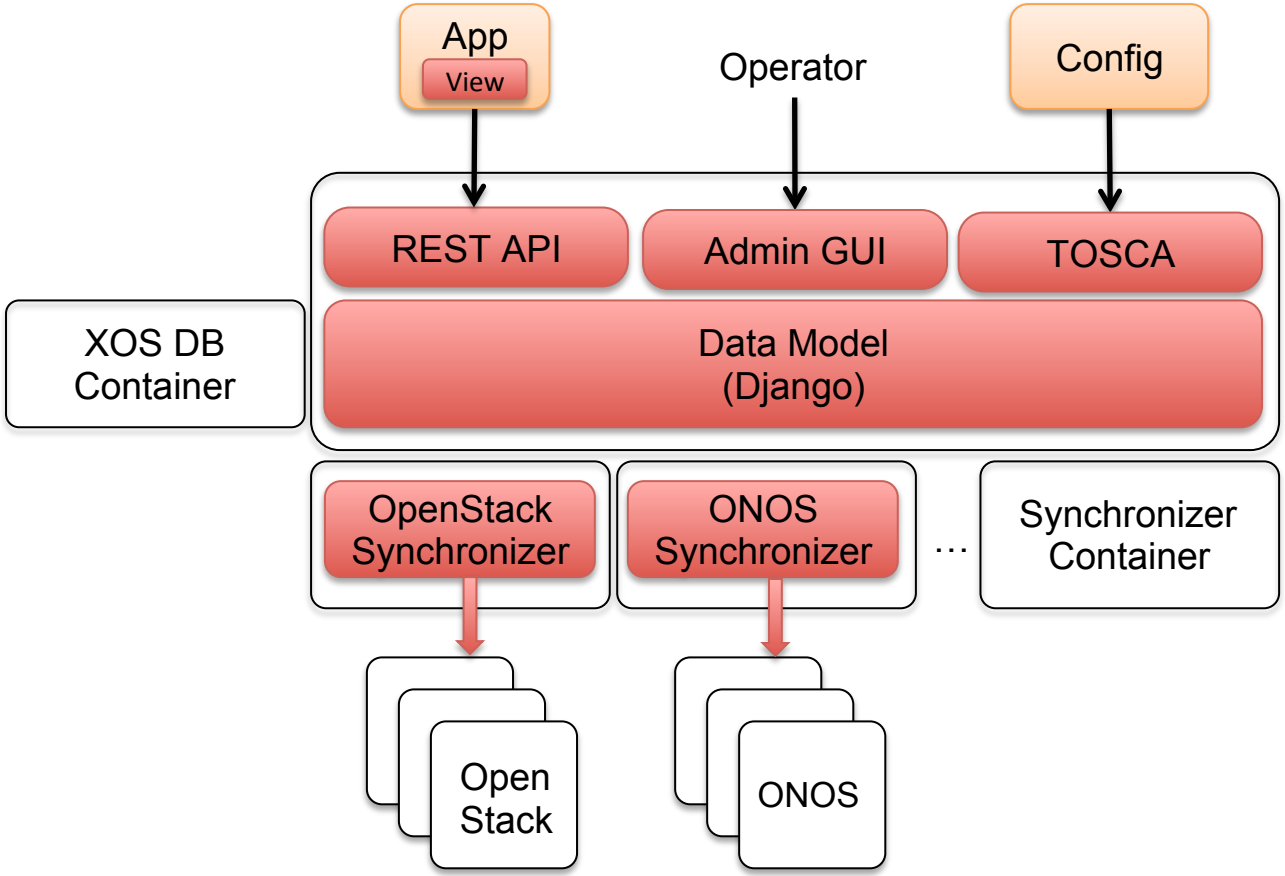
(2) CORD Controller mediates (and makes explicit) all inter-service dependencies

(3) CORD Controller provides a uniform interface to set of Service Controllers

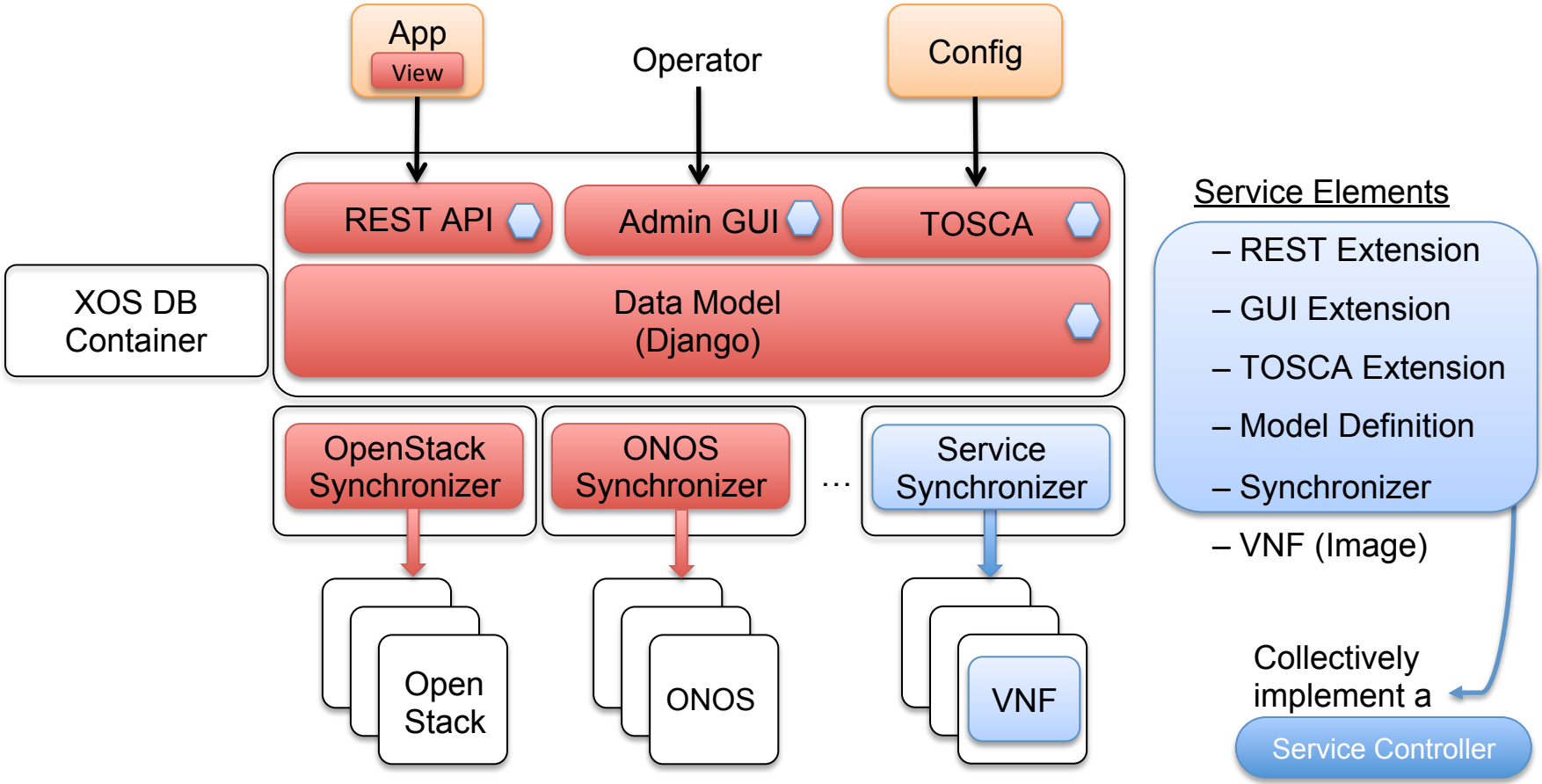
CORD Controller



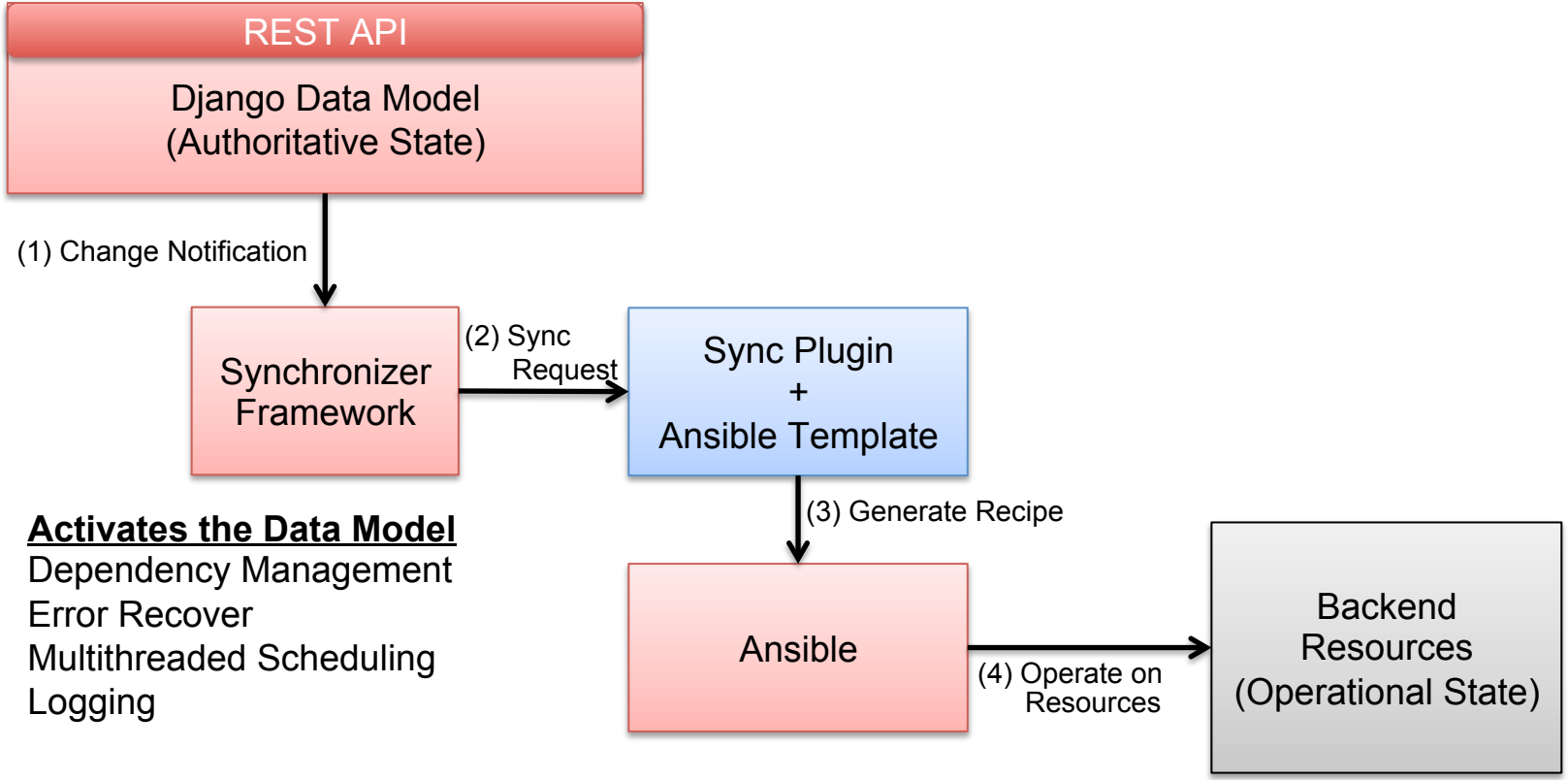
XOS Internals



Assembling a Service



Synchronizer and State Management



Activates the Data Model

Dependency Management
Error Recover
Multithreaded Scheduling
Logging

Service Elements

Component	Source Code for Service "X"
Django Model	**xos/services/X/model.py**
Synchronizer	xos/synchronizers/X/X-synchronizer.py xos/synchronizers/X/X_config **xos/synchronizers/X/steps/sync_X.py** **xos/synchronizers/X/steps/X_playbook.yml**
Admin GUI	xos/services/X/admin.py
REST API	xos/api/service/X.py xos/api/tenant/Xtenant.py
TOSCA Model	xos/tosca/custom_definitions/X.yaml xos/tosca/resources/X.py
Test/Documentation	xos/tests/api/hooks.py xos/tests/api/source/service/X.md xos/tests/api/source/tenant/Xtenant.md

On-Boarding a Service

Step	Action
Upload	Add files to github repository
Register	Edit `xos/settings.py` Edit `tools/xos-manage`
Configure	`cd xos/configurations/cord-pod` Edit TOSCA (`*.yaml`) files
Build	`make containers` `make`
Instantiate (optionally)	Create Service via Admin GUI Bind Service to Slice via Admin GUI Bind Slice to VNF Image

Runtime Execution – Subscriber API

GET http://portal.cord.net:9999/api/tenant/cord/subscriber/1/

Response:


```
{
    "humanReadableName": ”John Doe"
    "id": 1
    "features": {
        "cdn": true
        "uplink_speed": 4000000000
        "downlink_speed": 1000000000
        "uverse": true
        "status": "enabled"
    }
    "identity": {
        "account_num": "123"
    }
    "related": {
        "instance_name": "mysite_vcpe"
        "vsg_id": 4
        "c_tag": "432"
        "instance_id": 1
        "wan_container_ip": null
        "volt_id": 3
        "s_tag": "222"
    }
-
}
```

Runtime Execution – Subscriber API

PUT http://portal.cord.net:9999/api/tenant/cord/subscriber/1/features/uplink_speed

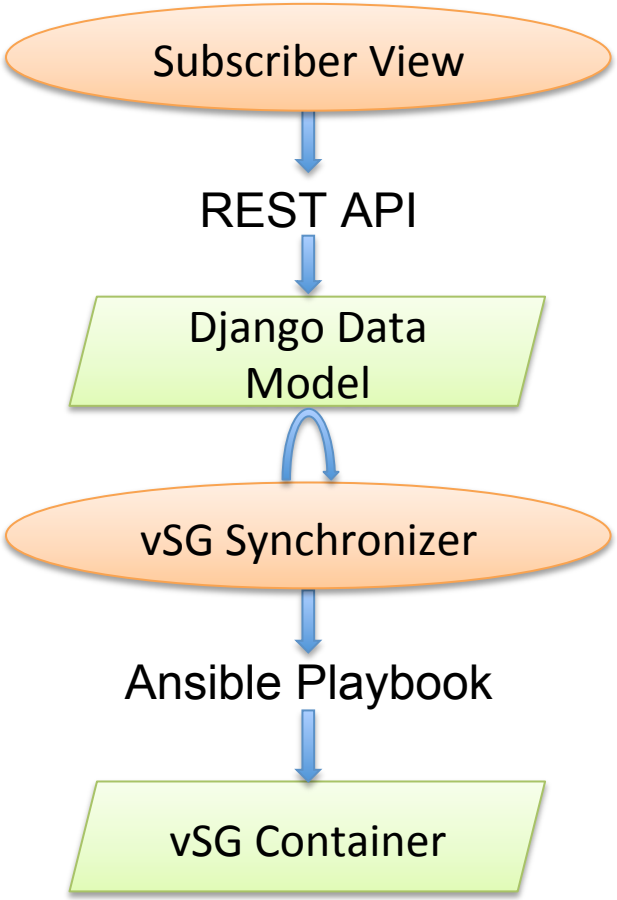
```
{
    "uplink_speed": 8000000000
}
```

Response:

```
{
    "uplink_speed": 8000000000
}
```

Documented at: http://docs.xos.apiary.io/

End-to-End Control Flow

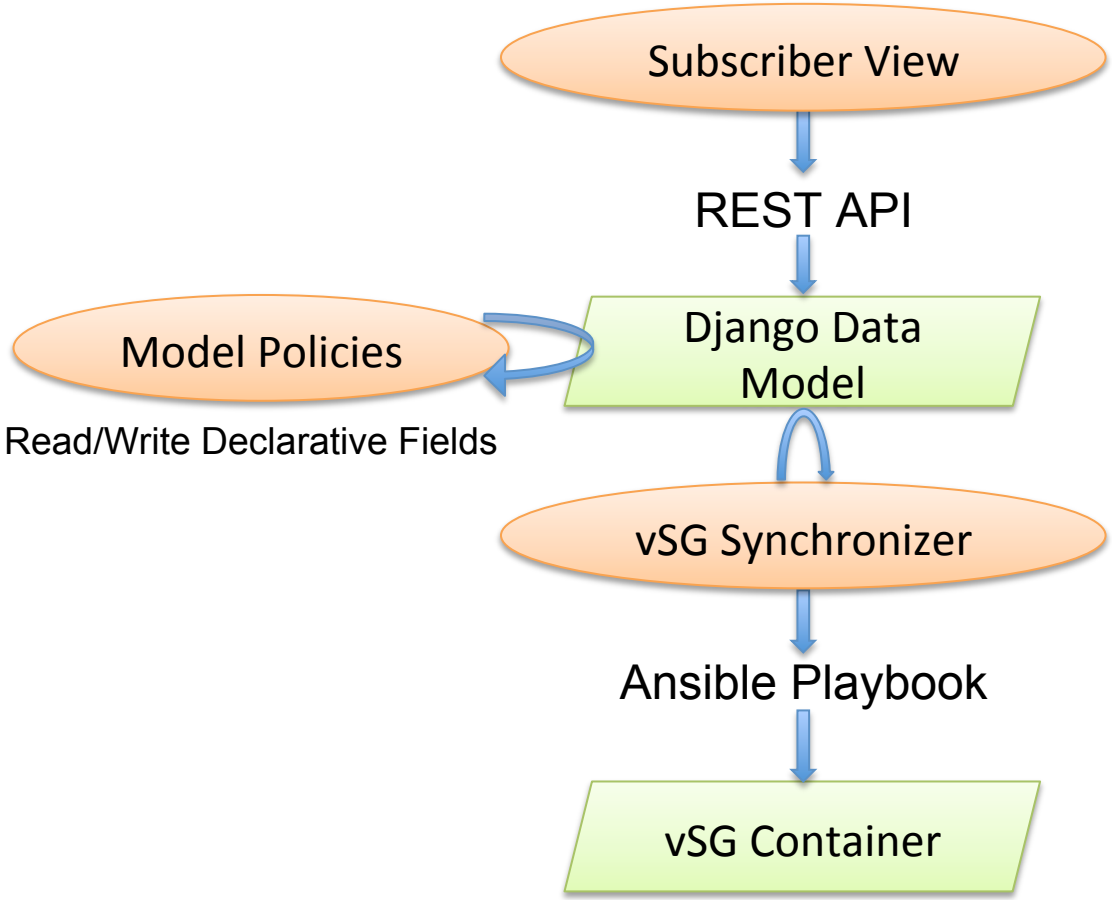


Read/Write Declarative Fields
Read Feedback Fields

Declarative Fields
Feedback Fields

Read Declarative Fields
Write Feedback Fields

Data Model “Maintenance”



Diagnostic Tools (1)

Nagios

LogStash

```
from xos.logger import logger
…
context = service.tologdict()
context['error_type'] = 'network'
logger.error("Could not connect to network", extra=context)
```

Diagnostic Tools (2)

Ceilometer

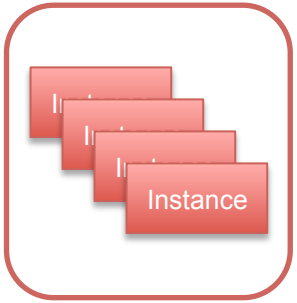
```
from kombu.connection import BrokerConnection
from kombu.messaging import Producer
…
connection = BrokerConnection(rabbit_host, rabbit_user,
             rabbit_password)
channel = connection.channel()
service_exchange = Exchange("myservice_xyz_rabbit_exchange",
             "topic", durable=False)
producer = Producer(channel, exchange=service_exchange,
             routing_key='notifications.info')
msg = …
producer.publish(msg)
```

Tutorial Outline

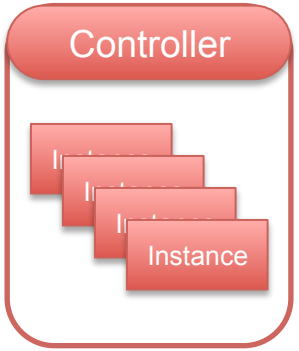
- Data Model
  - Core Models
  - Extending the Service Model
  - Modeling Services

Core Models

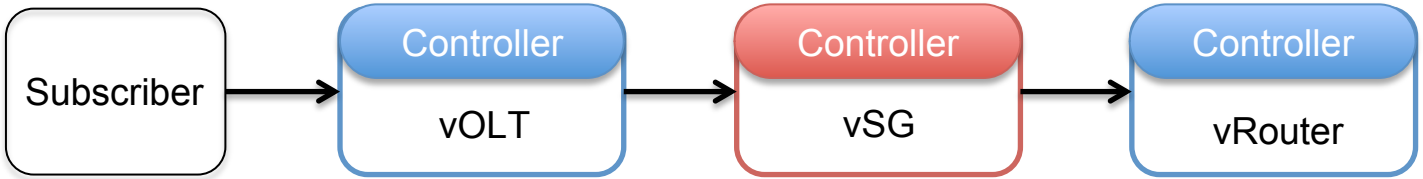
Instance =
(VM | Container |
Container-in-VM)



Slice =
(Instances[ ] + Networks[ ])



Service =
(“Controller” + Slices[ ])



“Service Graph” =
(TenantRoot + Services[ ] + Tenants[ ])

Core Models – Details

Model	Key Fields
Instance	(ID, Name, Isolation, Flavor, Image, Node, Ports[ ])
Isolation	VM | Container | Container-in-VM
Network	(ID, Name, Template, Parameters[ ], Links[ ])
Template Visibility Translation Access Topology Parameter	(ID, Name, Bandwidth, Visibility, Translation, Access, Topology) Public | Private NAT | None Direct | Indirect BigSwitch | Physical | Custom (Name, Value)
Port/Link	(Network, Instance, IP, MAC)
…	Nodes, Sites, Deployments, Users, Controllers…

Service-Related Class Hierarchy

Service

- → ONOSService
- → VOLTService
- → VROUTERService
- → VSGService
- → ExampleService

Tenant

- → ONOSApp
- → VOLTTenant
- → VROUTERTenant
- → TenantWithContainer
  - → VSGTenant
  - → ExampleServiceTenant
- → ServiceTenant

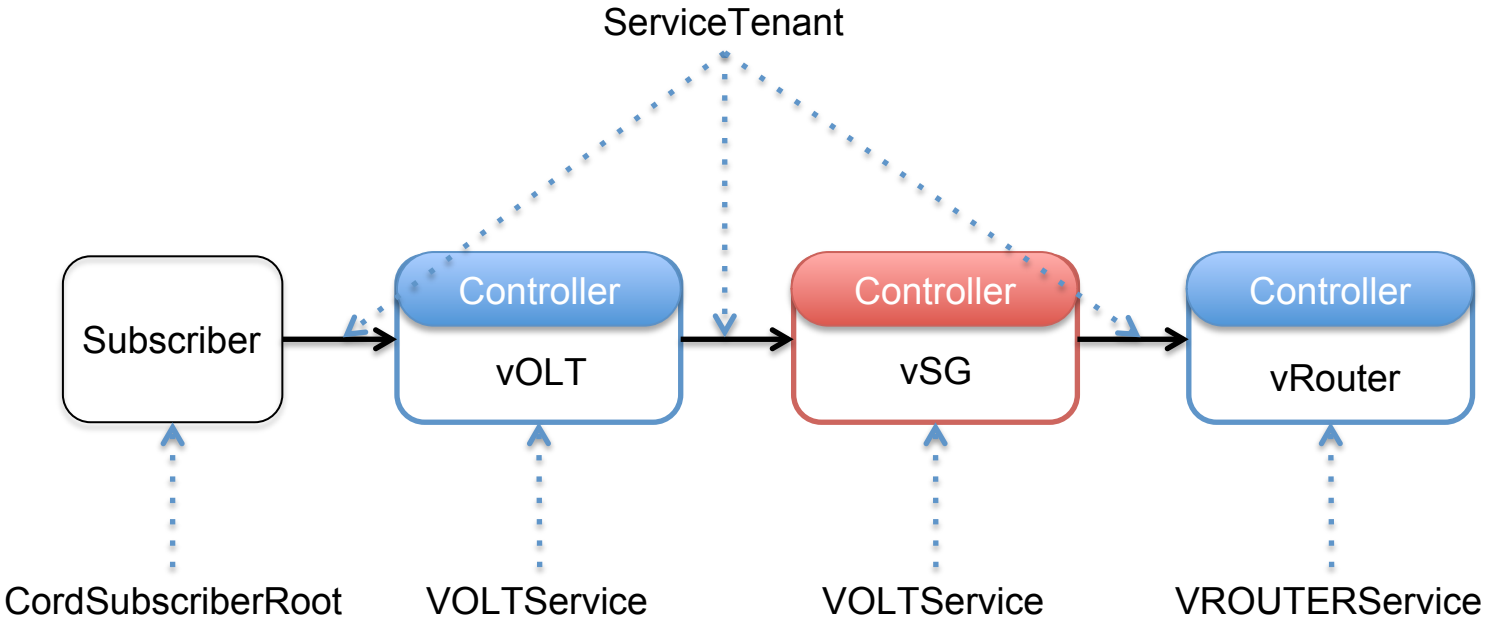
TenantRoot

- → Subscriber → CordSubscriberRoot
- → Provider

Service-Related Models – Details

Model	Key Fields
Service	(ID, Name, Kind, Slices[ ], Published, Version, Attributes[ ], Scale)
Attribute	(Key, Value)
Tenant	(ID, Name, Kind, Provider, Subscriber, ConnectMethod)
Provider Subscriber ConnectMethod	Service Service | TenantRoot | User | Tenant Public | Private | NA
TenantRoot	(ID, Name, Kind, Tenant)

Service Graph



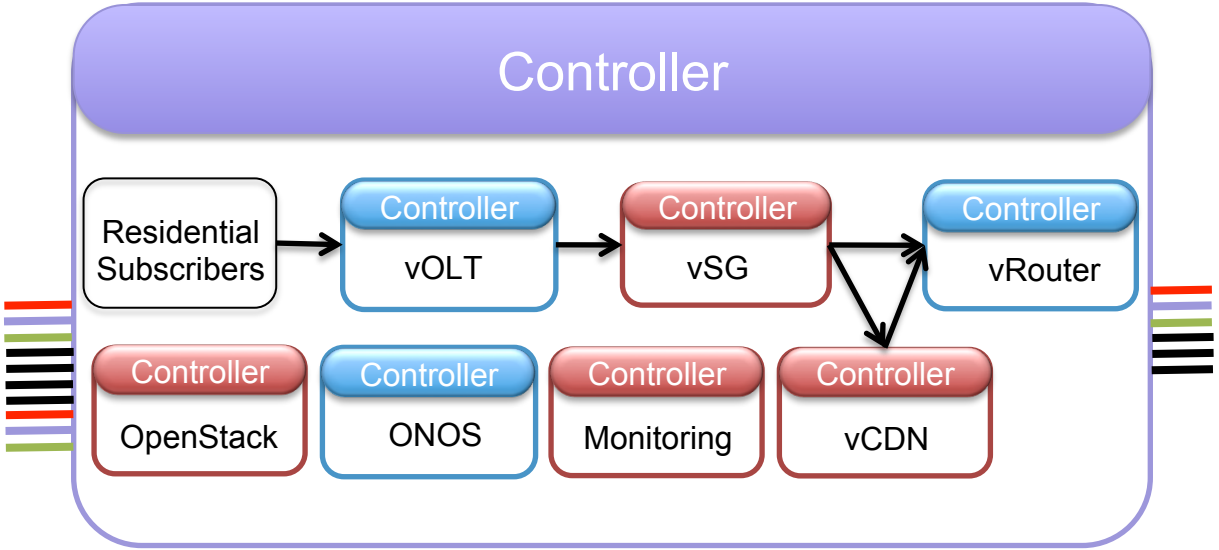
Modeling Services

Access-as-a-Service
and
Software-as-a-Service

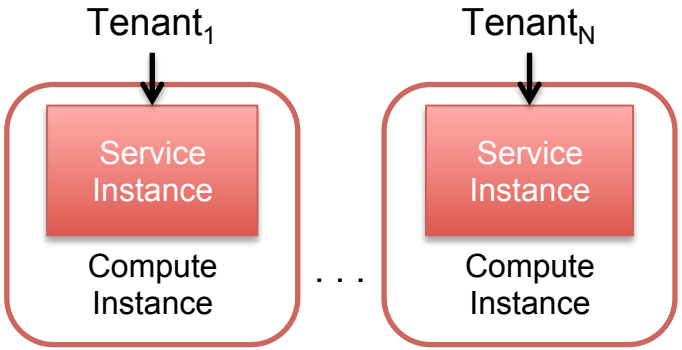
Bundled Legacy
and
Disaggregated
Greenfield

Data Plane (NFV)
and
Control Plane (SDN)

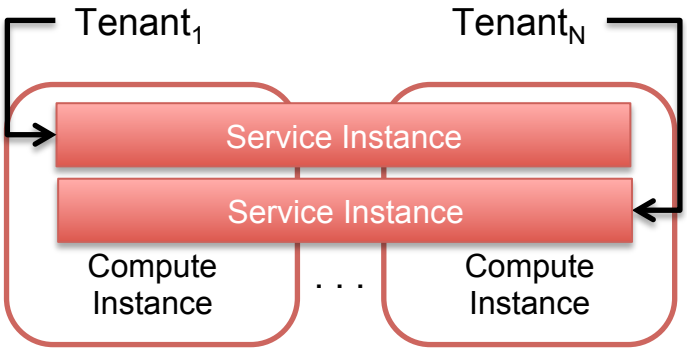
Trusted
and
Untrusted
3rd Party



Modeling – Scale and Isolation

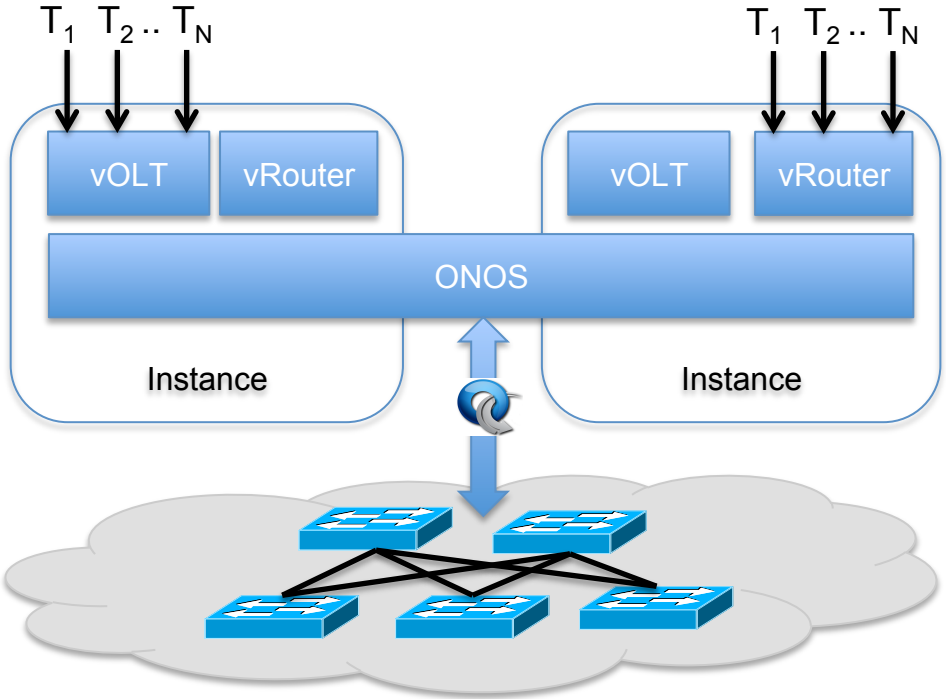


vSG
(Compute Instance = Container)

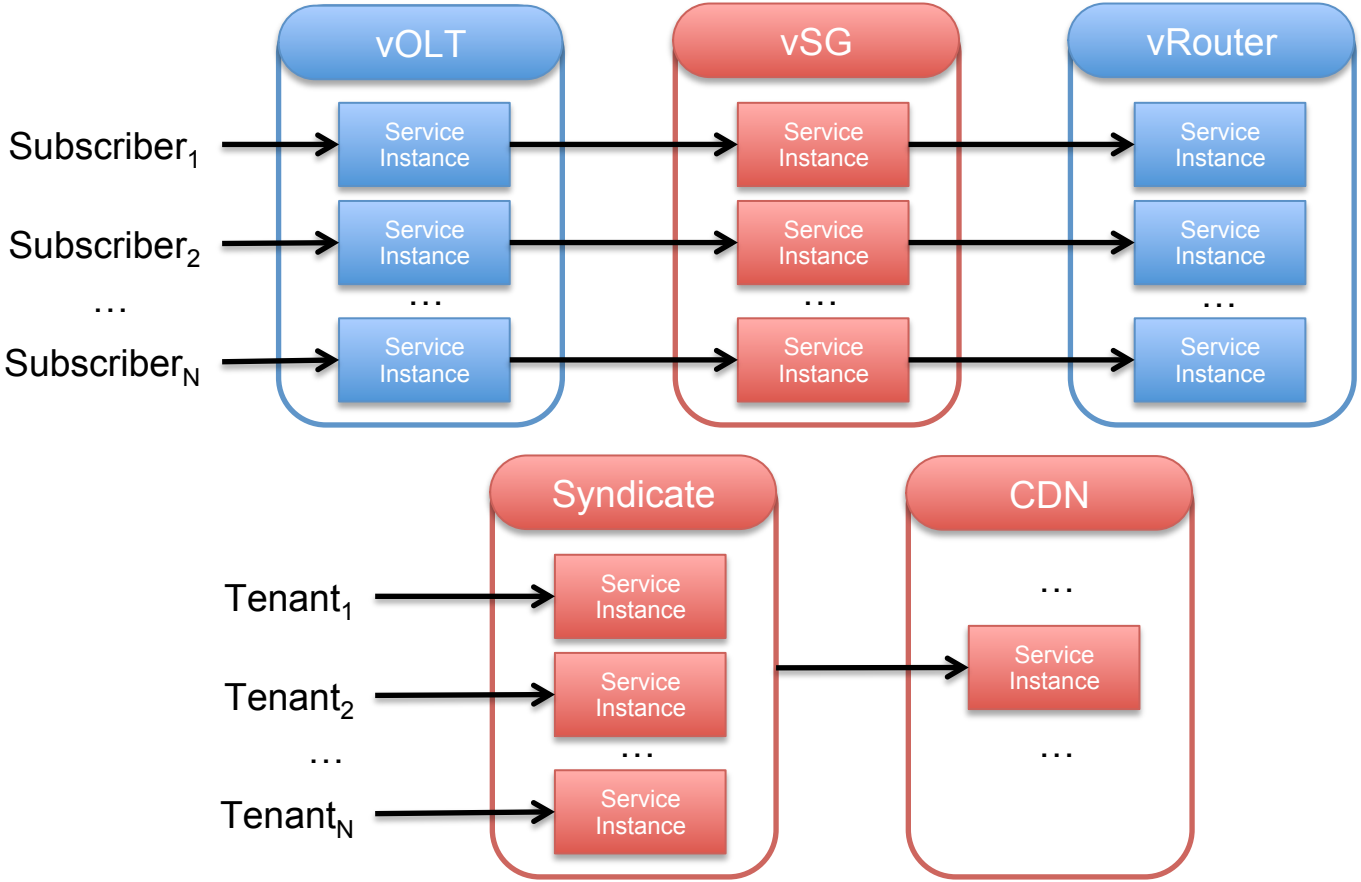


vCDN
(Compute Instance = VM)

Modeling – Scale and Isolation



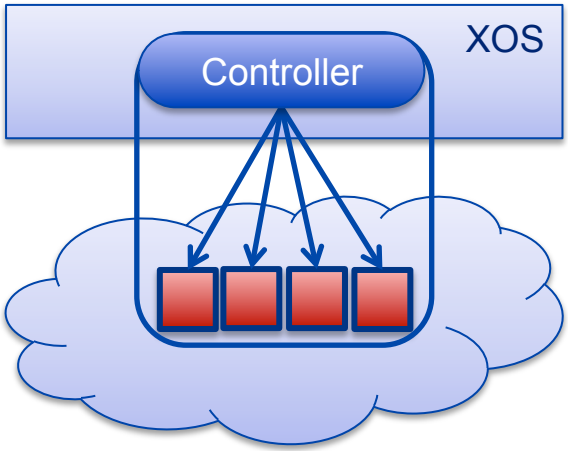
Modeling – Services as Tenants



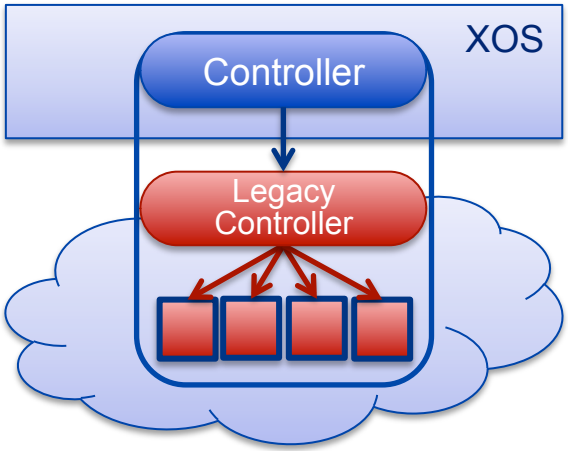
Modeling – From Native to Legacy

Blue – XOS Defined/Managed

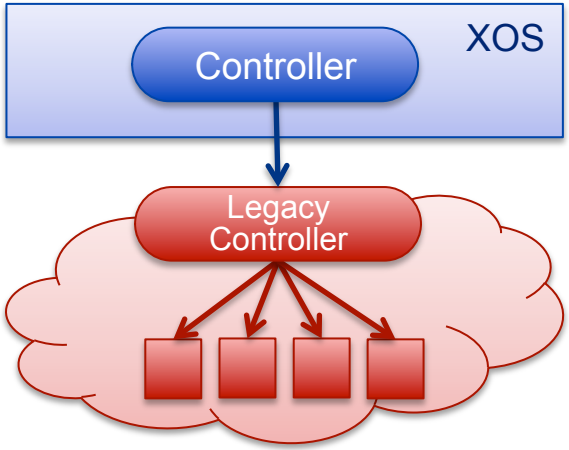
Red – External to XOS



e.g., vSG
(XOS includes tools to help construct a service)



e.g., vCDN
(XOS provides a means to coordinate VM acquisition & service init)



e.g., S3
(XOS provides a means to compose with an external service)

Tutorial Outline

- Assembling a Service
  - Defining a Model
  - Defining a Synchronizer
  - Extending APIs
    - REST
    - Admin
    - TOSCA
    - View

ExampleService – model.py

```
from core.models import Service, TenantWithContainer
from django.db import models, transaction

SERVICE_NAME = 'exampleservice'
SERVICE_NAME_VERBOSE = 'Example Service'
SERVICE_NAME_VERBOSE_PLURAL = 'Example Services'
TENANT_NAME_VERBOSE = 'Example Tenant'
TENANT_NAME_VERBOSE_PLURAL = 'Example Tenants'

class ExampleService(Service):

    KIND = SERVICE_NAME

    class Meta:
        app_label = SERVICE_NAME
        verbose_name = SERVICE_NAME_VERBOSE

    service_message = models.CharField(max_length=254, help_text="Service Message to Display")
```

ExampleService – model.py

```
class ExampleTenant(TenantWithContainer):
    KIND = SERVICE_NAME
    class Meta:
        verbose_name = TENANT_NAME_VERBOSE

    tenant_message = models.CharField(max_length=254, help_text="Tenant Message to Display")

    def __init__(self, *args, **kwargs):
        exampleservice = ExampleService.get_service_objects().all()
        if exampleservice:
            self._meta.get_field('provider_service').default = exampleservice[0].id
        super(ExampleTenant, self).__init__(*args, **kwargs)

    def save(self, *args, **kwargs):
        super(ExampleTenant, self).save(*args, **kwargs)
        model_policy_exampletenant(self.pk)

    def delete(self, *args, **kwargs):
        self.cleanup_container()
        super(ExampleTenant, self).delete(*args, **kwargs)
```

Model Policies

xos/services/exampleservice/model.py

```
def model_policy_exampletenant(pk):
    with transaction.atomic():
        tenant = ExampleTenant.objects.select_for_update().filter(pk=pk)
        if not tenant:
            return
        tenant = tenant[0]
        tenant.manage_container()
```

Synchronizer – sync_exampletenant.py

class SyncExampleTenant(SyncInstanceUsingAnsible)

```
    provides = [ExampleTenant]
    observes = ExampleTenant
    request_interval = 0
    template_name = “exampletenant_playbook.yaml”
    service_key_name = "/opt/xos/synchronizers/exampleservice/exampleservice_private_key"
    ....

 # Get the attributes that are used by the Ansible template but are not
 # part of the set of default attributes.
 def get_extra_attributes(self, o):
        fields = {}
        fields['tenant_message'] = o.tenant_message
        exampleservice = self.get_exampleservice(o)
        fields['service_message'] = exampleservice.service_message
        return fields
```

Synchronizer – exampletenant_playbook.yaml

```
- hosts: "{{ instance_name }}"
  connection: ssh
  user: ubuntu
  sudo: yes
  gather_facts: no

  vars:
    - tenant_message: "{{ tenant_message }}"
    - service_message: "{{ service_message }}”

 roles:
    - install_apache
    - create_index
```

Synchronizer – exampletenant_playbook.yaml

```
# roles/create_index/tasks/main.yml
- name: Write index.html file to apache
document root
  template:
    src=index.html.j2
    dest=/var/www/html/index.html

# roles/create_index/templates/index.html.j2
ExampleService
 Service Message: "{{ service_message }}"
 Tenant Message: "{{ tenant_message }}"
```

Synchronizer – Model Policies

Role

- Define operational dependencies *between* Models
- Change to Model A triggers a change Model B

Implementation

- Core: xos/synchronizers/openstack/model_policies
- Services: xos/services/X/model.py

Other Examples

CDN

OpenStack

ONOS

vOLT

vSG

Extending APIs

REST API

- xos/api/services/exampleservice.py
- xos/api/tenants/exampletenant.py
- xos/tests/api/source/service/exampleservice.md
- xos/tests/api/source/tenant/exampletenant.md

Admin GUI

- xos/services/exampleservice/admin.py

TOSCA

- xos/tosca/resources/exampleservice.py

Views

- views/README.md (tutorial video available)

Tutorial Outline

- Configuration Management
  - Elements of a Configuration
  - Canned Configurations
  - ExampleService Demo

Configuration Management

Look at:

- xos/configurations/frontend (developing XOS GUI)
- xos/configurations/devel (XOS development and testing)
- xos/configurations/cord-pod (contributing to CORD)

Bring up XOS containers (database, GUI, Synchronizers)

Configure XOS to talk to OpenStack backend

Add initial objects to XOS data model via TOSCA

Elements of a Configuration

Bring up XOS containers (database, GUI, Synchronizers)

docker-compose.yml – container definitions

Makefile – targets for manipulating containers

Configure XOS to talk to specific OpenStack backend

../common/Makefile.[cloudlab | devstack]

Add initial objects to XOS data model via TOSCA

*.yaml (other than docker-compose.yml)

Makefile – targets for invoking TOSCA engine

CORD-POD Configuration (1)

Walks through the process of bringing up a CORD POD

1. Install OpenStack on a single node or a cluster
   - Single node – no OLT or fabric, for development
   - Cluster – full POD, can integrate with OLT and fabric
2. Set up the ONOS VTN app (OvS on the nova-compute nodes is controlled by VTN)
3. Set up external connectivity for VMs
   - Handle case where there is no fabric
4. Bring up XOS with the CORD services

CORD-POD Configuration (2)

Head node “cheat sheet”

- *virsh list*: show all VMs created by install process
- *ssh ubuntu@<name>:* ssh to VM <name>

In XOS VM *(xos/xos/configurations/cord-pod)*

- *make*: Initial XOS setup and configuration
- *make vtn*: Generate VTN config and push to ONOS
- *make cord*: Configure CORD services and a sample household
- *docker-compose ps*: Show running containers
- *docker exec –ti <name> bash*: Enter container <name>

ExampleService Demo

https://github.com/open-cloud/xos/blob/master/xos/configurations/cord-pod/README-Tutorial.md

Set up a single-node CORD POD and configure VTN

Install all CORD services as well as ExampleService

Create a sample subscriber with a simulated device

Show that the subscriber can access the ExampleService

ExampleService Demo

To recap:

- Subscriber device sends HTTP GET request
- Device's packets have s-tag and c-tag added (sim OLT)
- Packets arrive on the VTN dataplane interface (sim fabric)
- In vSG instance: strips s-tag and c-tag, forwards to container
- vSG container forwards request to ExampleService instance
- Apache in ExampleService instance sends reply
- Reply takes the reverse path

A template for adding subscriber-facing services to CORD