Developing Services: Modeling and the XOS Toolchain

Scott Baker, scottb@opennetworking.org
Sapan Bhatia, sapan@opennetworking.org

CORD Build Nov. 7-9, 2017

An Operator Led Consortium


Goals of this Talk

- “I’m a VNF developer, what do I need to know to integrate my VNF into CORD ? How does CORD help my VNF interact with other VNFs?”
- “How can I make use of XOS’s generative toolchain to reduce the amount of manual coding that I need to do?”
- “What opportunities are there for me to contribute useful services or features to the CORD community?”

Modeling, Why is it Important?

- The design of CORD, XOS in particular, is driven by the data model
- Modeling is how…
  - … you make your service available on a CORD pod
  - … services learn about each other
  - … your service exposes configuration to the operator
  - … your service learns about resources (nodes, etc) on the pod
- This talk will
  - Show you how to model your service
  - Introduce you to the tools that interact with models

Modeling Basics, A Brief Overview

- XOS Data Model is Centralized
  - xos-core provides DataModel-as-a-Service, served via gRPC API
- XOS Data Model is Extensible
  - Services can register their own models
- XOS Data Model is Relational
  - Data is stored once and linked when necessary
  - Integrity / Constraints
    - Entity Integrity - no duplicates
    - Domain Integrity - restrictions on type, format, and range of values
    - Referential Integrity - data in use cannot be deleted

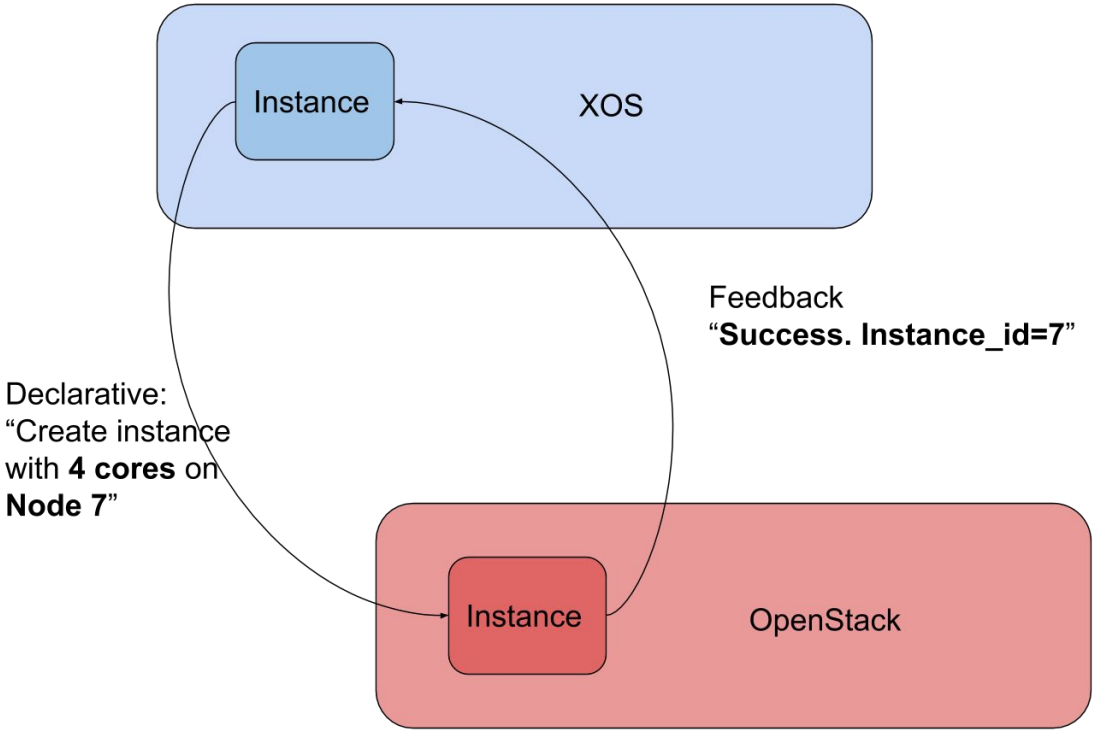
Modeling Constructs

- Models (aka tables)
  - Model = description of a data structure (aka schema)
  - Instantiated as objects (aka rows)
  - Primary key uniquely identifies an object
- Fields in a Model
  - Integer, Float, Text, Boolean, ...
  - Foreign Key - link from one object to another object
  - Computed - generated by using a query
    - Often used for reverse references to foreign keys
      - Slice.site / Site.slices

Declarative vs Feedback State

- Models have two types of members: Declarative & Feedback
- Declarative state
  - Generally specified by the operator
    - Example: name, bandwidth limit, network topology
  - Pushed from XOS to components to configure them
- Feedback state
  - Status information / identity information
    - Success | Fail, Error text
    - Component-specific ids (Example: instance_id, keystone_user_id)
  - Pulled from components to XOS

Declarative vs Feedback State

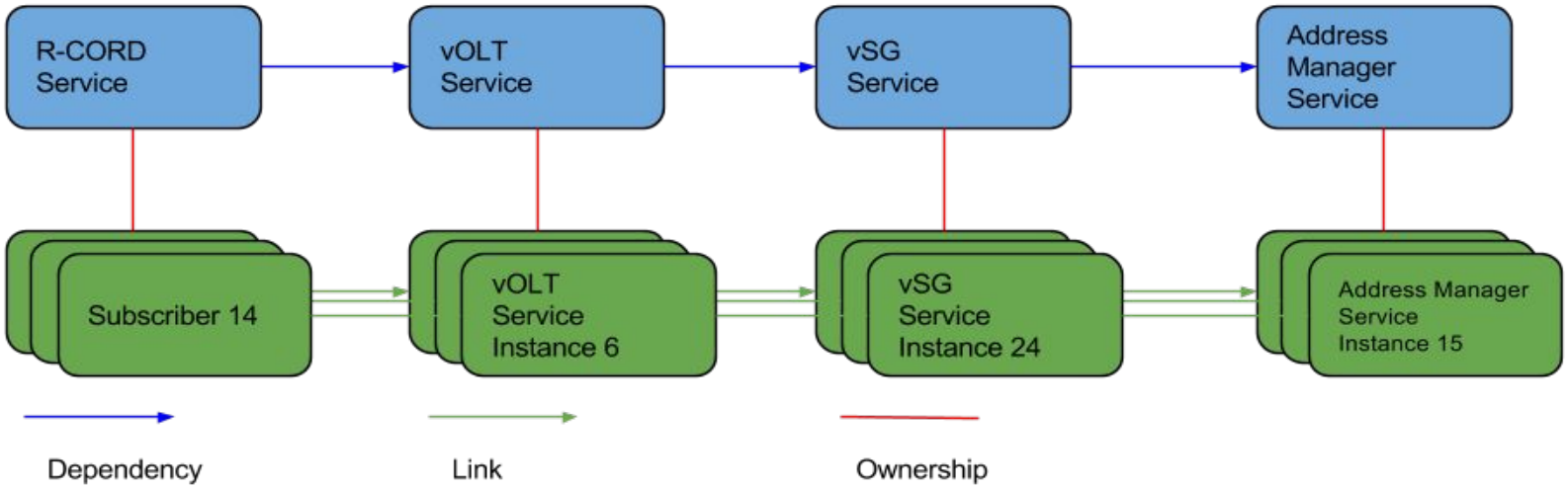


What Makes Up a Service ?

- Service Models
  - Service-wide / “global” parameters
- ServiceInstance Models
  - Divide a service into tenant-size pieces (“unit of tenancy”)
    - Example: Subscriber, ONOSApp, CDN ContentProvider
  - Not all Services are multi-tenant
- Auxiliary Models
  - In case everything doesn’t fit in a single model
  - Often when we need lists of items or items shared between ServiceInstances

Services and ServiceInstances

- One Service has many ServiceInstances



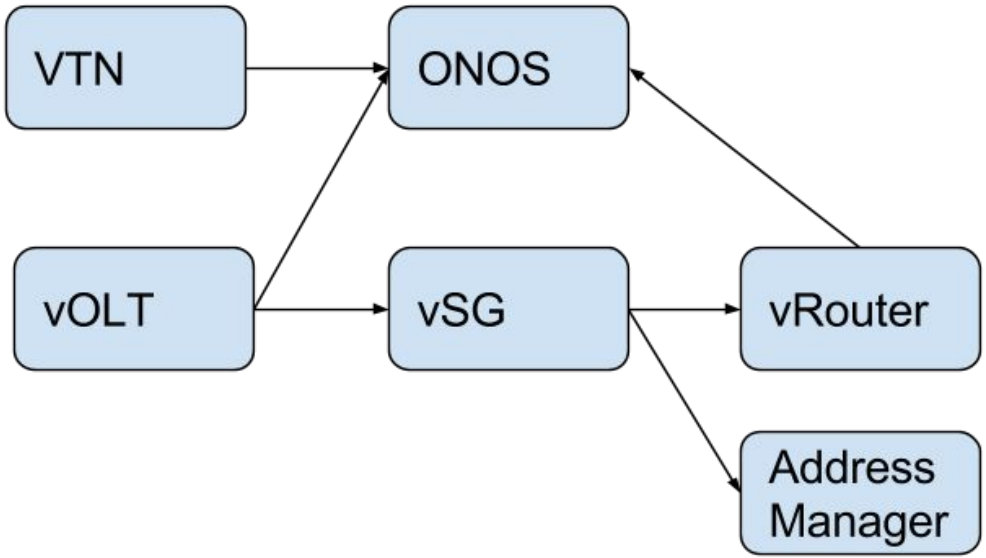
Connecting Services Together

- Services are often related to other services
  - Example: vSG has ingress and egress services
- XOS has modeling support for managing service graphs
  - Link models already exist; you don't need to create them
  - … but you should be aware how to use them.
- XOS implements two types of graphs
  - Service Dependency Graph
  - ServiceInstance Graph

Service Dependency Graph

- High level dependencies / requirements between Services
  - “VTN depends on ONOS”
  - “VSG depends on AddressManager”
- Does not necessarily imply connectivity
- Directed acyclical graph
- ServiceDependency model
  - (subscriber_service, provider_service, connect_method)
  - connect_method = [dataplane | controlplane | none]

Service Dependency Graph Example

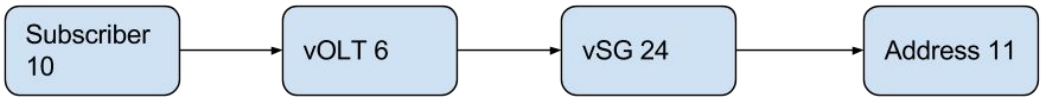


ServiceInstance Graph

- Express connections between ServiceInstances
  - Often implies data plane connectivity
- Arbitrary Graphs
  - Chains
  - Trees
  - DAGs
  - Cyclical Graphs
- ServiceInstanceLink Model
  - (subscriber_service_instance, provider_service_instance)

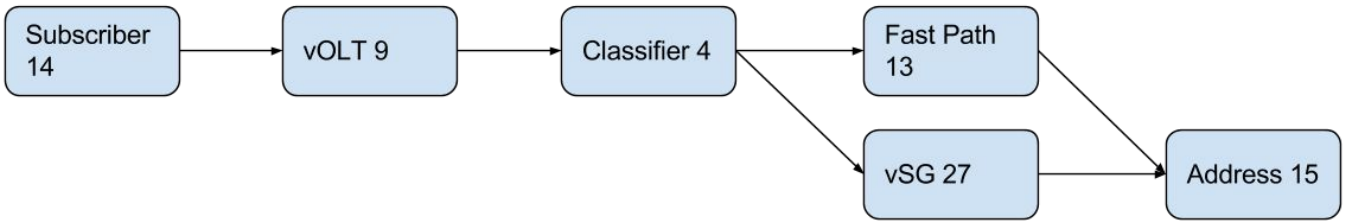
ServiceInstance Graph Example - Simple Chain

- The R-CORD Service Chain
  - Each subscriber has at least one vOLT
  - Each vOLT has a vSG
  - Each vSG has a public address



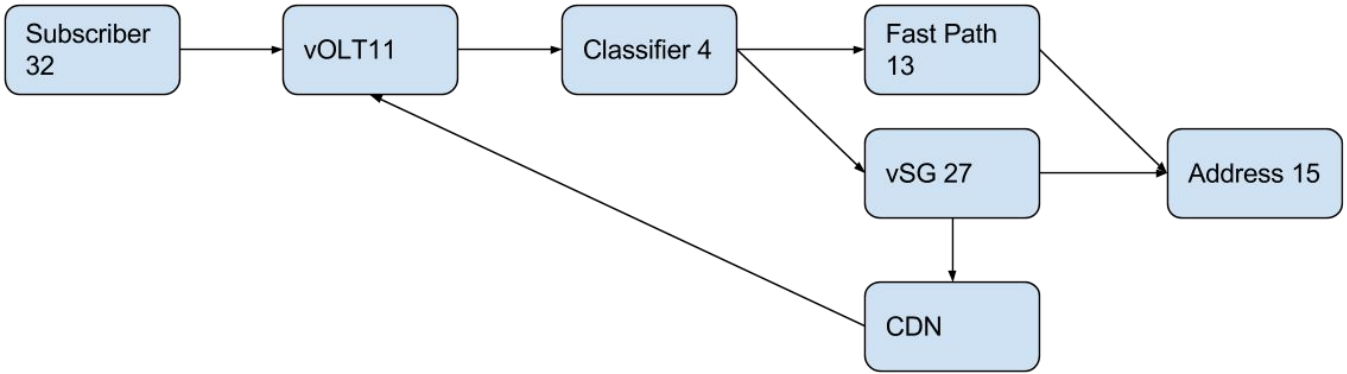
ServiceInstance Graph Example - Chain w/ Fork

- Let's extend the chain to include a classifier
  - Fast-path that could be implemented in ONOS
  - Slow-path that uses a vSG in a container



ServiceInstance Graph Example - Cyclical

- ServiceInstance graph supports cycles
  - Suppose there's an optimized CDN data path that could respond directly back to the client's access device



Introducing the XOS Generative Toolchain

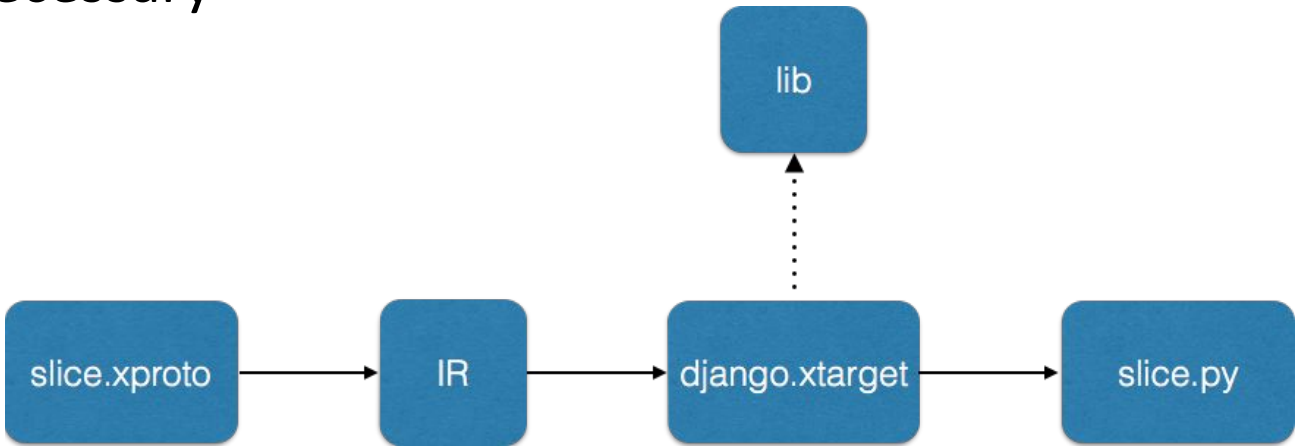
- Now that you know what models comprise a service…
- … let's dive into the specification language and the toolchain
- Services have many moving parts
  - Let's autogenerate them all from a single specification (xproto)
  - Developers' lives are easier when manual tasks are eliminated
    - Fewer mistakes
    - Less repetition / faster implementation
    - Better maintainability
    - More time spent on the interesting parts of the service

xproto, the XOS Modeling Language

- Requirements
  - Specify relations between models
  - Specify security policies
  - Specify validation policies
  - Support inheritance
- Based on Google Protobuf syntax
  - Protobuf message = XOS model
- 1:1 correspondence between xproto and protobuf + options
- xosgenx tool is the xproto translator

xosgenx

- xproto + jinja2 xtarget ⟶ output
  - xosgenx is inherently extensible to new use cases
- Automatically invoked by other tools (i.e. corebuilder) as necessary



Autogeneration Targets

- xosgenx generates many targets:
  - Database Schema
  - gRPC API / REST API (Chameleon)
  - TOSCA Schema
  - Graphical User Interface
  - Synchronizer Stubs
  - Documentation
  - Security Framework
  - …
- https://github.com/opencord/xos/tree/master/lib/xos-genx/xosgenx/targets

xosgenx Jinja2 Template Example

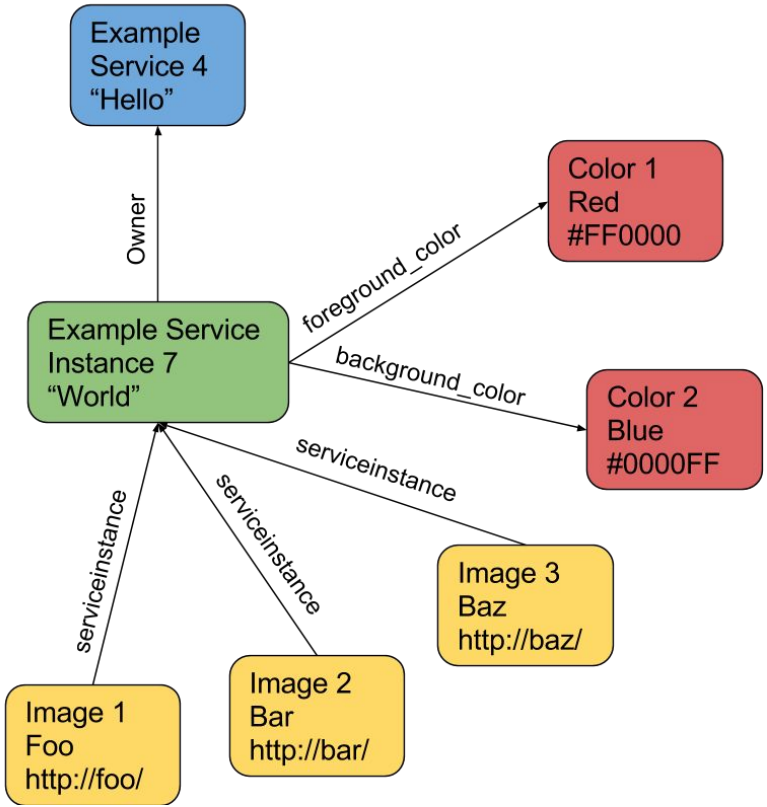
```
# dot.xtarget - generate a model graph


digraph {
{% for m in proto.messages %}
  {%- for l in m.links %}
  {{ m.fqn }} -> {{ l.peer.name }};
  {%- endfor %}
{% endfor %}
}
```

Let’s take a closer look at the xproto language

- To illustrate xproto, let’s walk through the modeling of a simple service that runs a web server
  - Unit of Tenancy = Web Server
- Service model
  - Global service-wide message string that appears in all servers
- ServiceInstance model
  - Tenant-specific message string (different for each server)
  - Foreground color, background color
  - Embedded images

ExampleService Models



ExampleService Service Model

```
message ExampleService (Service) {
      option verbose_name = "Example Service";
      required string service_message = 1 [help_text = "Service Message to
Display", max_length = 254, null = False, db_index = False, blank = False];
}

# (inherited from the core)
message Service (XOSBase) {
      required string description = 1 [help_text = "Description of Service",
max_length = 254, null = False, db_index = False, blank = False];
      required bool enabled = 2 [help_text = "Enable Service", default=True,
null = False, db_index = False, blank = True];
      ...
}
```

ExampleService Color Model

```
message Color (XOSBase){
     option verbose_name = "Color";
     required string name = 1 [help_text = "Name for this color", db_index =
False, max_length = 256, null = False, blank = False];
     required string html_code = 2 [help_text = "Code for this color",
db_index = False, max_length = 256, null = False, blank = False];
}
```

ExampleService ServiceInstance Model

```
message ExampleServiceInstance (TenantWithContainer){
     option verbose_name = "Example Service Instance";
     required string tenant_message = 1 [help_text = "Tenant Message to
Display", max_length = 254, null = False, db_index = False, blank = False];
     optional manytoone
foreground_color->Color:serviceinstance_foreground_colors = 2 [db_index =
True, null = True, blank = True];
     optional manytoone
background_color->Color:serviceinstance_background_colors = 3 [db_index =
True, null = True, blank = True];
}
```

ExampleService EmbeddedImage Model

```
message EmbeddedImage (XOSBase){
     option verbose_name = "Embedded Image";
     required string name = 1 [help_text = "Name for this image", db_index =
False, max_length = 256, null = False, blank = False];
     required string url = 2 [help_text = "URL for this image", db_index =
False, max_length = 256, null = False, blank = False];
     optional manytoone
serviceinstance->ExampleServiceInstance:embedded_images = 3 [db_index =
True, null = True, blank = True];
}
```

Policy Framework

- Limited logic expressions
- Two inputs
  - Security context (current user, calling API type, etc)
  - Entire data model
- Output
  - True | False
- Automatically compiled into python
  - General purpose, can be used outside XOS

xproto Policies

```
# validation policies
policy exampleservice_validator < (obj.service_message in [“hello”, “goodbye”]) >
policy exampleserviceinstance_validator < (obj.tenant_message in [“world”, “planet”]) >
# security policies
policy exampleservice_policy <ctx.user.is_admin | exists Privilege: Privilege.accessor_id =
ctx.user.id & Privilege.accessor_type = "User" & Privilege.object_type = "ExampleService" &
Privilege.object_id = obj.id >
policy exampleserviceinstance_policy < *exampleservice_policy(obj.owner) >
# usage
message ExampleService::exampleservice_policy (Service) {
    option validators = “exampleservice_validator:ExampleService does not have allowed
service_message”
```

When does xosgenx run?

- xosgenx is run automatically by the build system
  - Typically during corebuilder phase
- xosgenx can also be run manually
  - For iterative development / testing
  - Generate service stubs (data model, sync, policies)
- We suggest you attend the hands-on tutorial session
  - You will learn how to deploy a service
- Let’s move on to talk about exploring the data model with xossh...

xossh

- Shell that allows interaction with data model
  - Connects via gRPC API to xos core
- Used for debugging/development
  - Inspection of existing data model
  - Create, modify, or delete objects in data model
- Running xossh from CORD headnode
  - `/opt/cord/orchestration/xos/xos/tools/xossh -u xosadmin@opencord.org -p <password>`
- Live demo

Opportunities for Community Help

- Grow the inventory of CORD services
  - New VNFs
    - Example: firewall
  - New support services
    - Example: Cassandra as a Service
- Expand autogeneration use cases
  - More xtarget files
  - Example: automate VNF implementation or syncstep generation

Resources

- CORD Guide:
  - http://guide.opencord.org/

Beyond End-of-Talk

- Slides beyond this point are extra content not intended to be presented unless needed

RDBMS Table Example

Sites Table

id	name	Description
7	mysite	This is my site

Users Table

id	email	site_id	name
4	scottb@opennetworking.org	7	Scott Baker

xossh demo 1 - login

```
vagrant@head1:~$ /opt/cord/orchestration/xos/xos/tools/xossh -u xosadmin@opencord.org -p
8wJdFjyHslKii74YQboV

__   __   ____      _____     _____   _     _
\ \ / /  / __ \    / ____|   / ____| | |   | |
 \ V /  | |  | | | (___     | (___   | |__| |
  > <   | |  | |  \___ \     \___ \  |  __  |
 / . \  | |__| |  ____) |    ____) | | |  | |
/_/ \_\  \____/  |_____/    |_____/  |_|  |_|


XOS Core server at xos-core.cord.lab:50051
Type "listObjects()" for a list of all objects
Type "listUtility()" for a list of utility functions
Type "login("username", "password")" to switch to a secure shell
Type "examples()" for some examples
xossh >>>
```

xossh demo 2 - listing model types

```
xossh >>> listObjects()
['AccessAgent', 'AccessDevice', 'AddressManagerService', 'AddressManagerServiceInstance',
'AddressPool', 'AgentPortMapping', 'Controller', 'ControllerImages', 'ControllerNetwork',
'ControllerPrivilege', 'ControllerRole', 'ControllerSite', 'ControllerSitePrivilege',
'ControllerSlice', 'ControllerSlicePrivilege', 'ControllerUser', 'CordSubscriberRoot',
'Deployment', 'DeploymentPrivilege', 'DeploymentRole', 'Diag', 'ExampleService',
'ExampleServiceInstance', 'FabricService', 'Flavor', 'Image', 'ImageDeployments',
'Instance', 'InterfaceType', 'Network', 'NetworkParameter', 'NetworkParameterType',
'NetworkSlice', 'NetworkTemplate', 'Node', 'NodeLabel', 'ONOSApp', 'ONOSService', 'Port',
'Privilege', 'Role', 'Service', 'ServiceAttribute', 'ServiceDependency', 'ServiceInstance',
'ServiceInstanceAttribute', 'ServiceInstanceLink', 'ServiceInterface',
'ServiceMonitoringAgentInfo', 'ServicePrivilege', 'ServiceRole', 'Site', 'SiteDeployment',
'SitePrivilege', 'SiteRole', 'Slice', 'SlicePrivilege', 'SliceRole', 'Tag',
'TenantWithContainer', 'User', 'VOLTDevice', 'VOLTService', 'VOLTTenant', 'VRouterApp',
'VRouterDevice', 'VRouterInterface', 'VRouterIp', 'VRouterPort', 'VRouterService',
'VRouterTenant', 'VSGService', 'VSGServiceInstance', 'VTNService', 'VTRService',
'VTRTenant', 'XOS', 'XOSGuiExtension']
xossh >>>
```

xossh demo 2 - listing and dumping

```
xossh >>> ExampleServiceInstance.objects.all()
[<ExampleServiceInstance: exampletenant1>]

xossh >>> ExampleServiceInstance.objects.first().dump()
backend_code: 1
backend_need_delete: true
backend_need_delete_policy: false
backend_need_reap: false
backend_register: "{\"next_run\": 0, \"last_success\": 1509150998.978467, \"exponent\": 0}"
backend_status: "OK"
…
tenant_message: "world"
updated: 1509150868.64
write_protect: false
class_names: "ExampleServiceInstance,TenantWithContainer,ServiceInstance,XOSBase"
self_content_type_id: "exampleservice.exampleserviceinstance"

xossh >>>
```

xossh demo 3 - modifying

```
xossh >>> si = ExampleServiceInstance.objects.first()
xossh >>> si.tenant_message = "Some Other Message"
xossh >>> si.save()
xossh >>>
```

xossh demo 4 - creating and deleting

```
xossh >>> example_service = ExampleService.objects.first()
xossh >>> si = ExampleServiceInstance(tenant_message="a new tenant", owner =
example_service)
xossh >>> si.save()
xossh >>>


xossh >>> while si.backend_code == 0:
xossh ...     si = ExampleServiceInstance.objects.get(id = si.id)
xossh ...
xossh >>> print si.backend_code, si.backend_status
1 OK
xossh >>>


xossh >>> si.delete()
xossh >>>
```

xossh demo - script

```
/opt/cord/orchestration/xos/xos/tools/xossh -u xosadmin@opencord.org -p <password>

listObjects()

ExampleServiceInstance.objects.all()
ExampleServiceInstance.objects.first().dump()

si = ExampleServiceInstance.objects.first()
si.tenant_message = "Some Other Message"
si.save()

example_service = ExampleService.objects.first()
si = ExampleServiceInstance(tenant_message="a new tenant", owner = example_service)
si.save()

while si.backend_code == 0:
    si = ExampleServiceInstance.objects.get(id = si.id)
print si.backend_code, si.backend_status

si.delete()
```

XPROTO syntax example

- Model declaration
  - ```
    message Slice::slice_policy (XOSBase) {
    …
    }
    ```
- String field
  - ```
    required string name = 1 [max length = 80, content type = "stripped", blank = False, help_text = "The Name of the Slice", null = False, db_index = False];
    ```
- Foreignkey field
  - ```
    required manytoone site->Site:slices = 6 [help text = "The Site this Slice belongs to", null = False, db_index = True, blank = False];
    ```
    ```

xosgenx Jinja2 Template Examples (2)

```
# synchronizer.xtarget - generate a synchronizer config file


name: {{ app_label | lower }}-synchronizer
accessor:
  username: xosadmin@opencord.org
  password: "@/opt/xos/services/{{ app_label | lower }}/credentials/xosadmin@opencord.org"
dependency_graph: "/opt/xos/synchronizers/{{ app_label | lower }}/model-deps"
steps_dir: "/opt/xos/synchronizers/{{ app_label | lower }}/steps"
sys_dir: "/opt/xos/synchronizers/{{ app_label | lower }}/sys"
model_policies_dir: "/opt/xos/synchronizers/{{ app_label | lower }}/model_policies"
+++ {{ app_label | lower }}_config.yaml
{% endfor %}
```

Xosgenx Jinja2 Template Examples (3)

- Many more examples...
  - https://github.com/opencord/xos/tree/master/lib/xos-genx/xosgenx/targets
  - django.xtarget - generate django models
  - grpc_api.xtarget - generate grpc api
  - modeldefs.xtarget - generate modeldefs spec used by GUI
  - proto.xtarget - generate protobufs
  - swagger.xtarget - generate swagger documentation

Developer Tools

- General
  - Docker, Docker-compose
  - Elastic Stack
- Generative Toolchain
  - corebuilder
  - imagebuilder
  - xosgenx
  - xossh

Developer Tools: Docker / Docker-Compose

- Managing containers
  - List containers: `docker ls`
  - Go inside a container: `docker exec -it <container_id> bash`
  - Stop a container: `docker stop <container_id>`
  - Destroy a container: `docker rm <container_id>`
  - Pull changes from local repo: `docker-compose -p rcord pull`
  - Restarting containers: `docker-compose -p rcord up -d`

Developer Tools: Elastic Stack

- Used for collecting logging from xos components
  - Synchronizers
  - Xos core
- Supports querying and filtering log data

Developer Tools - Corebuilder

- Used for building the core container
  - Input is a set of onboarding recipes
  - Bakes data model from xproto specifications
  - Generates xos-core container image
- Static build-time tool
- Expected to be replaced by dynamic onboarding “soon”

Developer Tools - Imagebuilder

- Used for building container images during Build phase
- Labels container images with versioning information
- Uses versioning labels and tags to determine if a prebuilt image can be pulled from Docker Hub, or if it must be rebuilt locally