XOS: A SERVICE FOR DEPLOYING VPNS IN THE CLOUD

XOS: A SERVICE FOR DEPLOYING VPNS IN THE CLOUD

By

JEREMY DALE MOWERY

_____

A Thesis Submitted to The Honors College

In partial Fulfilment of the Bachelor's degree

With honors in

Computer Science and Mathematics

THE UNIVERSITY OF ARIZONA

MAY 2016

Approved by:

_____

Dr. John H. Hartman
Department of Computer Science

XOS: A SERVICE FOR DEPLOYING VPNS IN THE CLOUD

By

JEREMY DALE MOWERY

MAY 2016

Table of Contents

Abstract

XOS[1] is an Everything as a Service operating system designed for the modern cloud that uses the OpenStack platform. XOS makes it easy to create and deploy new cloud services. For this project I developed two new services for XOS, the first a tutorial service that is used as a framework for the current tutorial documentation for XOS, and a OpenVPN service that allows operators to create new VPNs using OpenVPN. The OpenVPN service is the subject of this document, it provides an easy to use interface for creating secure VPN servers and adding clients to VPNs. The benefit of using a VPN is security. All computers connected to a VPN can communicate privately and securely in isolation[2]. In many situations this is useful, such as database servers that should be isolated from a larger network. Future plans for the OpenVPN service are to support replication across multiple sites, adding entire slices as clients to a VPN, and adding an entire service as a client to a VPN. The research for this thesis came primarily from understanding XOS to create the tutorial service, and from understanding OpenVPN and its security requirements including managing a Public Key Infrastructure.

Introduction

Cloud Computing is a generic term that typical defines a wide range of services that run on computers and virtual machines accessed through the internet. As opposed to typical computing where services and applications run on a local machine, cloud services typical run over a distributed set of machines and virtual machines running in facilities around the world. Many consumer products, from storage solutions like Dropbox and Google Drive, to virtual machine hosting like AWS and Microsoft Azure, leverage the cloud to provide lightweight implementations of services for clients.

In the field of Cloud Computing there are several service models that define the services that are provided to users, these include Software as a Service (SaaS) in which users use software running on a cloud infrastructure, Platform as a Service (PaaS) through which users can deploy their own cloud services using the provided infrastructure, and Infrastructure as a Service (IaaS) in which the configuration and provisioning of resources is controlled by the user over a provided cloud infrastructure[3].

XOS is a operating system running on the cloud that uses an Everything as a Service (XaaS) model which combines the three above service models. XOS powers the OpenCloud project, a

---

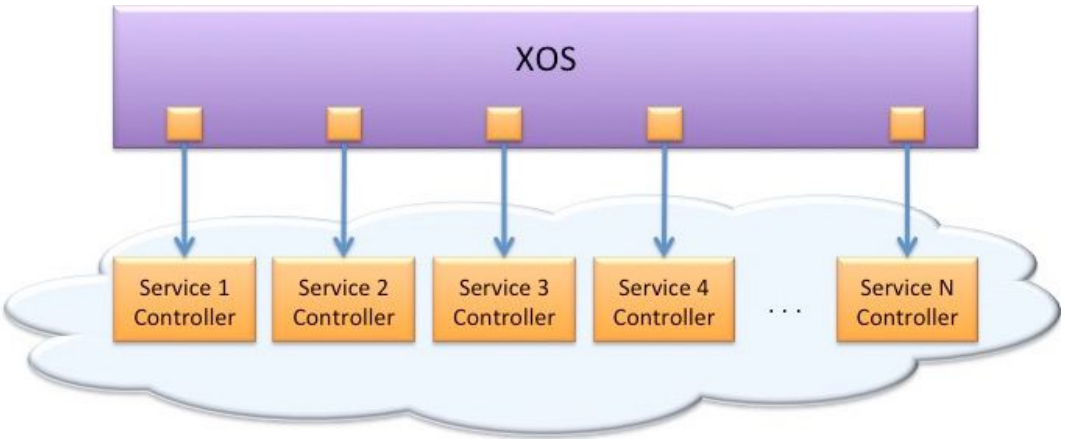
[1] "XOS Guide · XOS Guide." 2015. 23 Apr. 2016 <http://guide.xosproject.org/>

[2] "Why OpenVPN?." 2011. 23 Apr. 2016 <https://openvpn.net/index.php/open-source/335-why-openvpn.html>

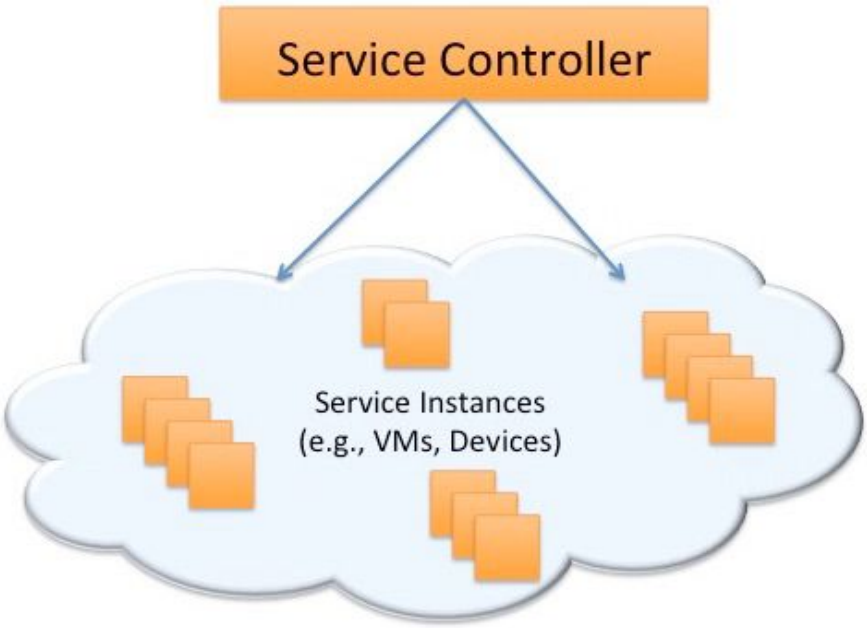
[3] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." 6 (2011): 20-23.

“showcase for innovative cloud services”, services added to XOS are made available through OpenCloud[1].

In XOS everything is implemented as a *Service* each of which has a *Service Controller* that provides the functionality for implementing the service across many *Service Instances* that run in virtual machines and devices in clusters. The following figures from the XOS Guide illustrate the design.



XOS as an abstraction for services defined by Service Controllers[4]

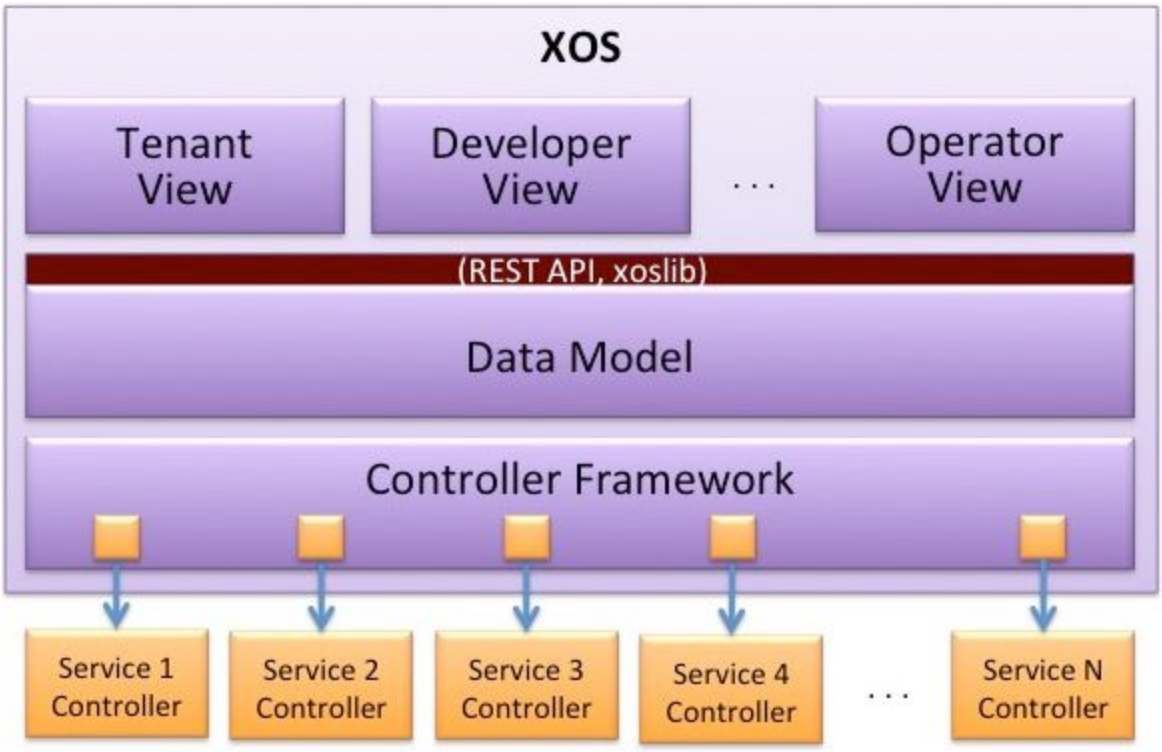


A Service Controller as an interface for interacting with VMs[4]

[4] "Architecture Guide · XOS Guide." 2016. 24 Apr. 2016 <http://guide.xosproject.org/archguide/>

From a programming perspective, *Services* are objects in a well-defined data model and are associated with collections, known as a *slices*, of VMs, known as *instances*. The management of instances in a slice is a Service in XOS, thus XOS implements a Tenancy Model to assist with the management of instances. Through this model *Tenants* are added to the data model, they represent the link between a Service and the creation of instances, all tenants have a reference to a provider service and contain functionality for managing instances of that provider service. *Synchronizers* are programs, part of the Controller Framework, that monitor changes to the Data Model and perform the enaction of Services and Tenants on instances. Synchronizers often use Ansible for scalable low level configuration.

The following block diagram from the XOS guide illustrates the software design of XOS, starting with views used by users to interact with XOS, the data model representing the Services and Tenants, the the Controller Framework with Service Controllers that enact the state of the data model using synchronizers[5].



A Block Diagram representing the structure of XOS[5]

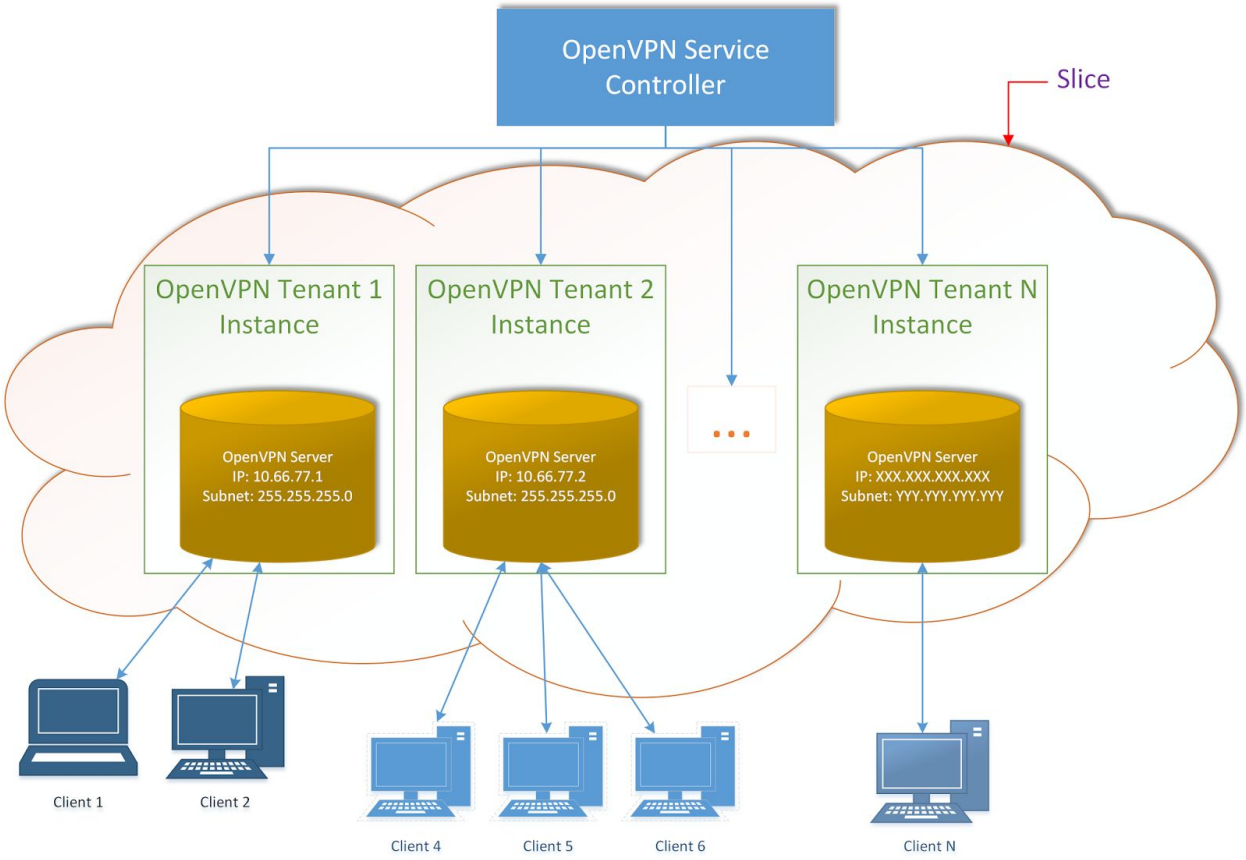
.

[5] "Architecture Guide · XOS Guide." 2016. 24 Apr. 2016 <http://guide.xosproject.org/archguide/>

One useful service that was missing from XOS was a VPN Service. A Virtual Private Network (VPN) is a network that exists in a virtual layer (ie without physical data connections) to isolate and encrypt network traffic between multiple computers.[6] VPNs have many uses, such as isolating networks handling sensitive information and adding additional security to everyday web traffic. A VPN service is useful to OpenCloud from two perspectives. From a user perspective such a service would allow users to create VPN servers to secure their own network traffic or join an existing network such as a University's network. From an administrative perspective a VPN service in OpenCloud would allow services themselves to be isolated. All instances in a service's slice could be added to a VPN to keep traffic between those instances secure. Furthermore, services that represent layers of a larger service, could be isolated so that intra-service traffic is isolated from the outside world. For example, for a web service composed of a file hosting service and a database service the traffic between the databases and the file hosting could be isolated from the rest of the Internet.

The lack of a VPN Service in OpenCloud has prevented this level of network traffic isolating and security. To solve this problem I created an XOS VPN Service that makes use of OpenVPN (an open source application to create VPN servers and connect clients) to start OpenVPN servers on instances. This service runs on OpenCloud utilizing the power of XOS to create instances in an associated slice that run the OpenVPN server, one server per instance. Using this new service, operators can be create new OpenVPN servers with customizable configurations including options for fault tolerance. The following diagram shows how the service works from an XOS perspective, servers are created inside instances and clients connect to those servers.

[6] "What is OpenVPN?." 2011. 23 Apr. 2016 <https://openvpn.net/index.php/open-source/333-what-is-openvpn.html>



OpenVPN Service diagram

For this thesis I implemented an OpenVPN service the implements the functionality described above. The following is a list of my contributions to the XOS project:

- Created the backend infrastructure to create OpenVPN servers
- Create a frontend UI for administrators to create an OpenVPN service with port allocation
- Created a frontend UI for users to create new OpenVPN Tenants (servers)
- Added an access control mechanism for Tenant Objects
- Created a frontend UI for users of an OpenVPN Tenant to download a script to connect to a particular OpenVPN server
- Created a HelloWorldService example that served as the basis of the current tutorial on creating services

User Guide

As a user of XOS you can create an OpenVPN Tenant that create a new VPN by starting a new OpenVPN server.

You can add users to your VPN who will then have access to a script to connect to it.

You may also remove users which will end their access to your VPN.

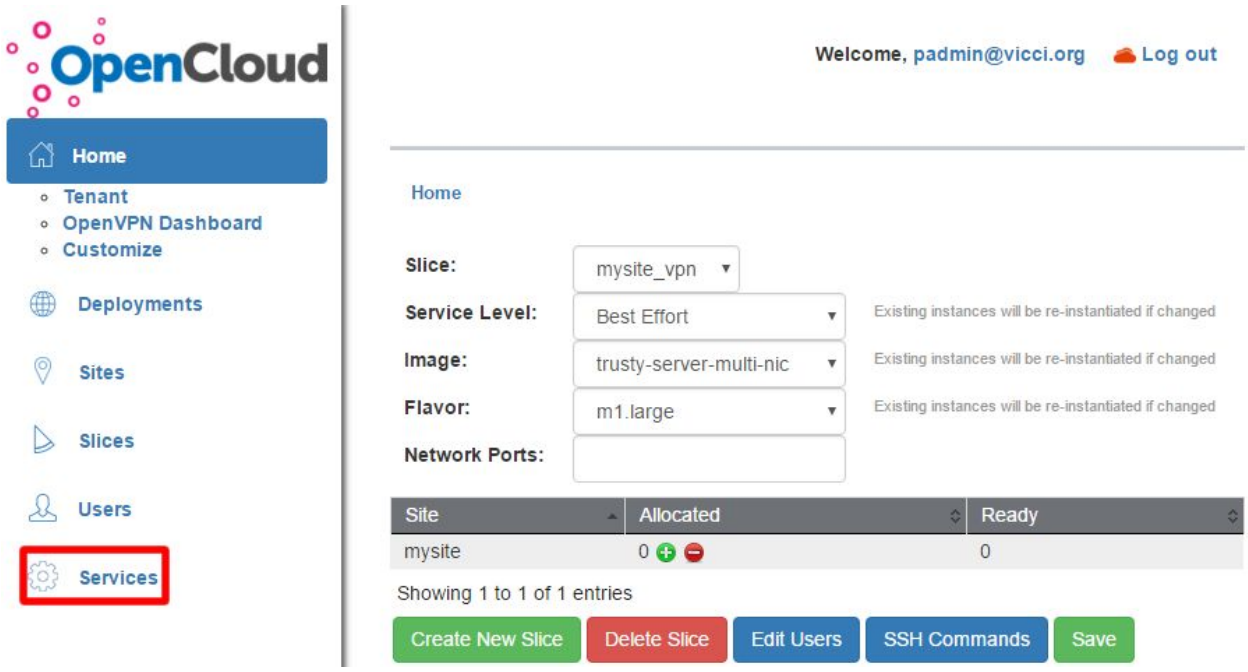
Creating an OpenVPN server by creating an OpenVPNTenant

To create an OpenVPN server you will need to create an OpenVPNTenant in XOS. You will do this by submitting a form with your configuration options. There is a one-to-one relationship between OpenVPNTenants and OpenVPN servers (and VPNs).

Step 1: Navigate to the OpenVPNTenant add form

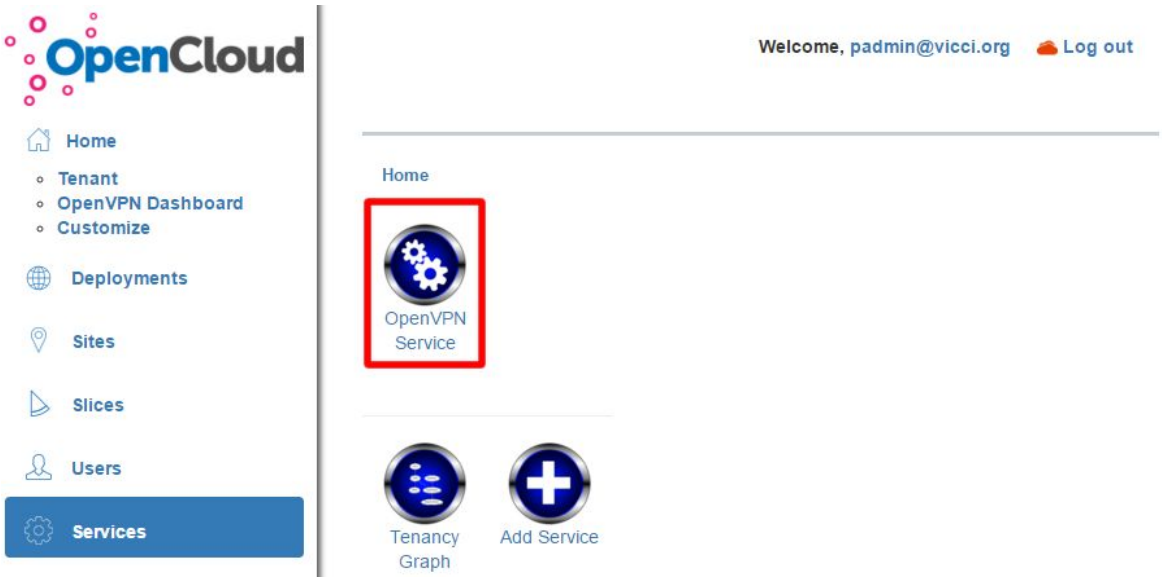
To add a new OpenVPNTenant you must navigate to the add form by doing the following:

1. In the left hand navigation bar click **Services**



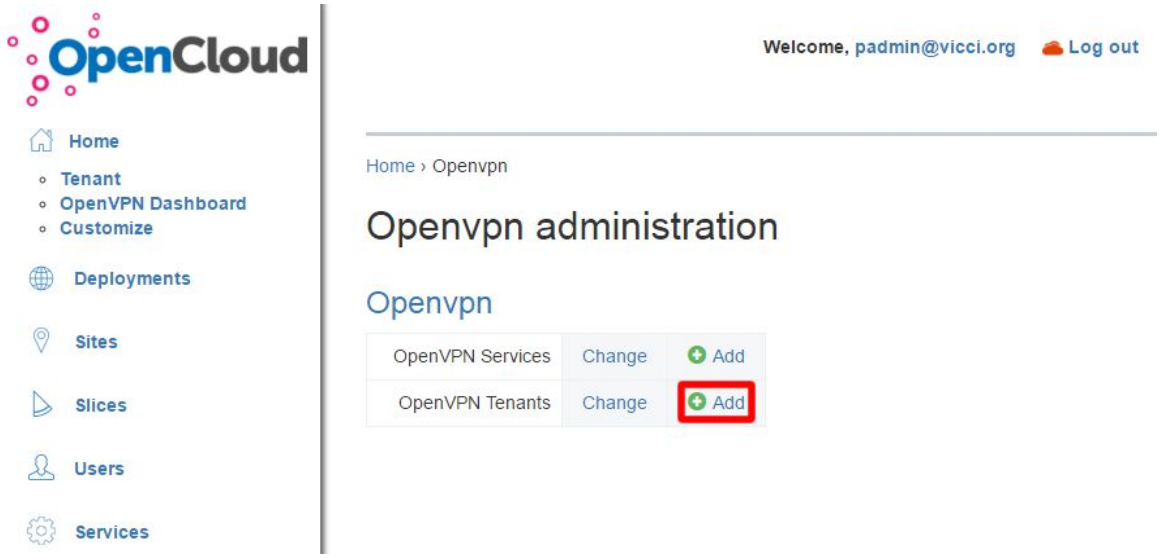
The Services button

2. In the middle section click the service with the name of your OpenVPN Service (you may need to ask your administrator what the name of this service is)



Choosing the OpenVPN Service

3. In the middle section, in the row labeled **OpenVPN Tenants** click **Add**



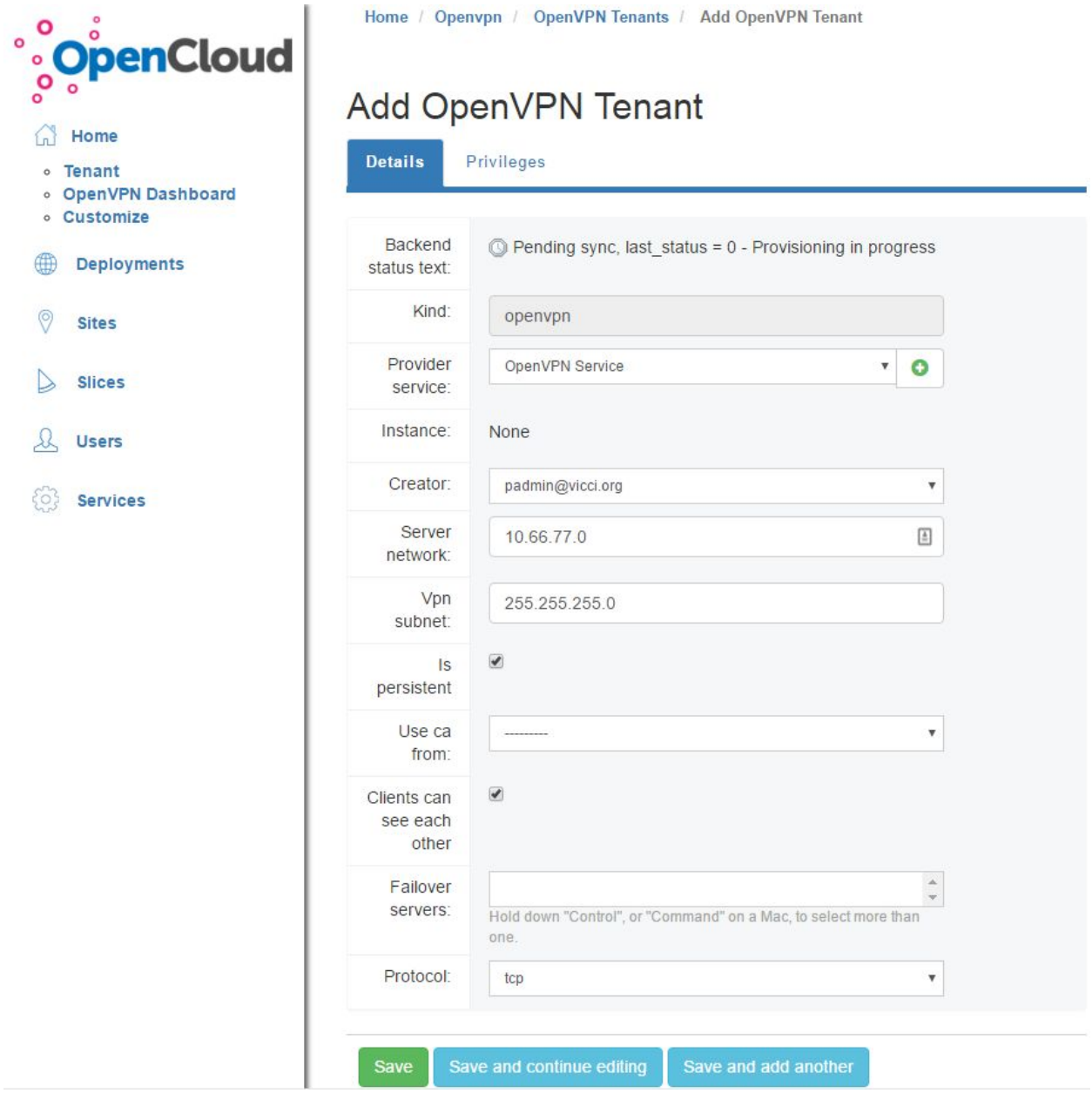
The add button

Step 2: Create your new OpenVPNTenant instance

Now that you are on the OpenVPNTenant form you can create a new OpenVPNTenant (and thus an OpenVPN server) by filling out the form and clicking **Save**. Below are the configuration options available listed under the tab they are found in, an asterisk denotes a required field.

Note that after clicking **Save** it maybe a few minutes before your server is available. You will know that your OpenVPN server is ready when the **Backend status text** shows a green checkmark with the text **"Successfully enacted"**

Details



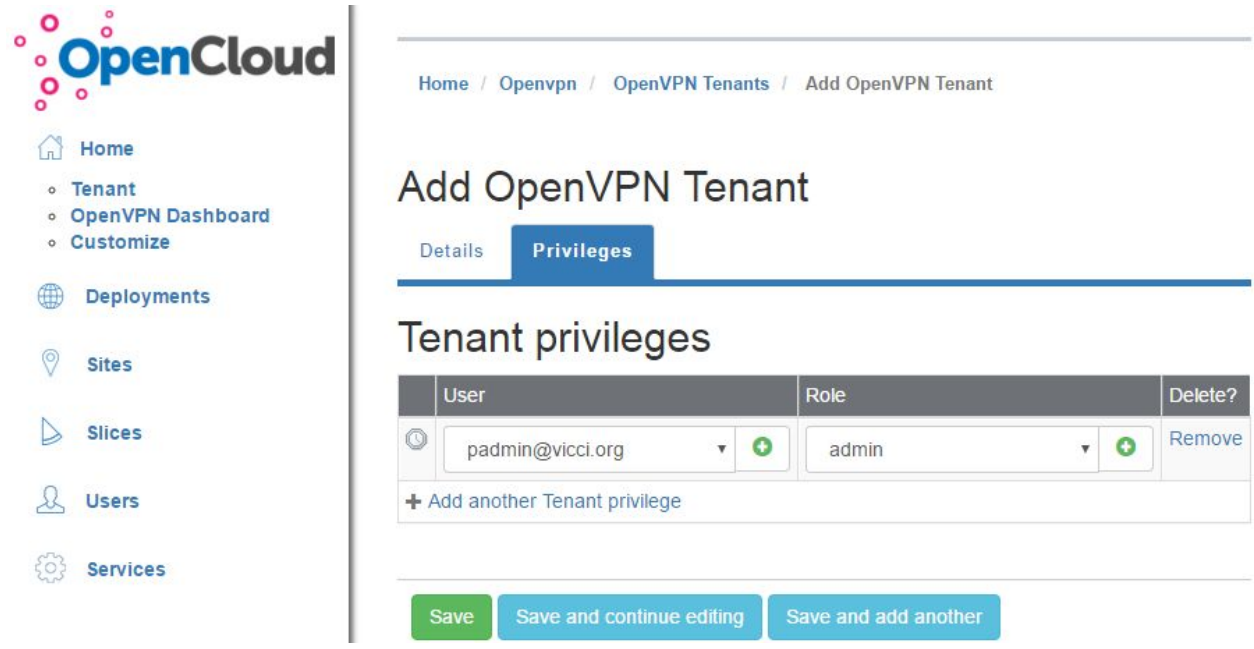
The default Add OpenVPN Tenant form

- **Provider Service***: The service that this OpenVPN Tenant is associated with. The OpenVPN server will listen on a port obtained from the Provider Service instance.
- **Creator***: The user who will be recorded as the owner of the OpenVPN Tenant.
- **Server Network***: The IPv4 address for the network used for the VPN for this Tenant (note that this is not the server address, it is the network that the server is on). Note that if you want clients to be connected to multiple VPN networks you will need to use a different server network for each. XOS does not enforce the uniqueness of the networks, so it will be up to you to decide the best network to use. OpenVPN recommends using subnets in 10.0.0.0/8 because the addressing space is large, the default network is 10.66.77.0 and some other examples include 10.88.99.0, 10.101.0.0, and 10.57.240.0
- **VPN Subnet***: The subnet for the network. This defines the network used for servers and clients. For example if you wanted to use the network 10.66.77.0/24 you would use the Servier Network above as 10.66.77.0 and the **VPN Subnet**, 255.255.255.0. The combination of the **Server Network** and **VPN Subnet** will define the number of addresses on the network, and therefore a maximum number of clients that can connect to the VPN.
- **Is Persistent**: Determines whether or not the OpenVPN server connects clients to the same IP address every time they connect and every time that the server restarts. Addresses are assigned from the network defined from the **Server Network** and **VPN Subnet** and are therefore limited. If this option is selected previously connected clients will receive the same address on the VPN every time, however, this means that if a client never connects again the address will not be free for other clients to use. Thus this option is useful if your network depends on machines have a static IP (a printer, a server, etc...) but not ideal if you have limited address space and clients that may never connect again.
- **Use CA From:** A OpenVPN Tenant that you wish to use the Certificate Authority (CA) from. Use this when you want to allow clients to connect to multiple VPNs using the same certificate. OpenVPN works by having certificates for clients signed by a CA, so by enabling this option you will have certificates for clients signed by the same CA as is used on multiple VPNs. This is a requirement for failover to work.
- **Clients can see each other**: If this option is enabled the clients of a OpenVPN server can communicate with each other over the VPN. If this option is disabled clients can only send packets to the OpenVPN server.

- **Failover servers:** A list of OpenVPN servers to use if one being created is unavailable or overloaded. Note for this to work all VPNs in the list must be using the same CA using the use ca from option.
- **Protocol***: The protocol to use for the server, TCP or UDP. UDP provides the best performance and security but the least data integrity, packets may be dropped between the server. Using TCP will ensure that packets are delivered to the OpenVPN server if they are dropped but decreases performance and makes the network more vulnerable to attacks that target TCP.

Note that if you are intending for clients to connect to multiple VPNs you will need to make sure that no two networks your clients connect to overlap. XOS does not currently do this automatically, thus you will need to manually decided on a network that will not overlap with other networks. OpenVPN explicitly recommends using subnetworks in the middle of the 10.0.0.0/8 block.

Privileges



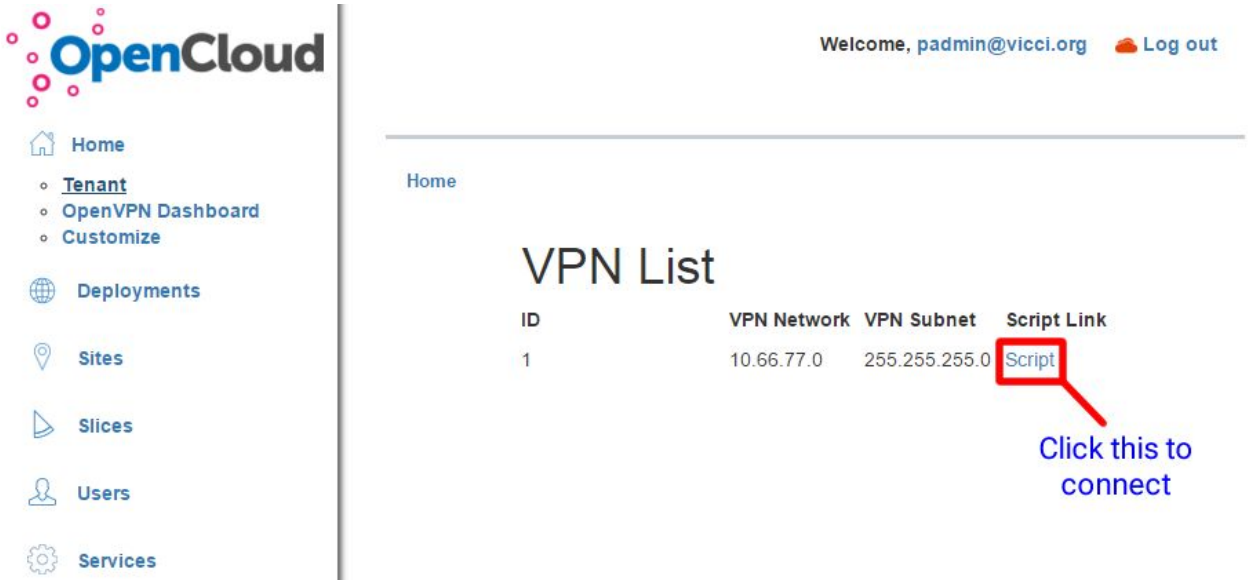
Adding user padmin@vicci.org as an admin user of this OpenVPN Tenant

Use the **privileges tab** to define the users that can access the VPN. Every user you add with any permission level can access the VPN via the OpenVPN Dashboard described below. At any point you may revoke a user's access to the VPN by using the checkbox in the **Delete**

column on the right (available after saving), when a user is deleted their certificate is revoked, and the OpenVPN server they are connected to will drop all traffic from the client.

Using the OpenVPN Dashboard to connect to a VPN

The OpenVPN Dashboard is used by users to connect to OpenVPN servers that they have been given access to. You can find the dashboard as a link in the left navigation area. The dashboard gives users a list of OpenVPN servers to which they can connect. If a user clicks on a **script link** they will download a script that when run will allow them to connect to the selected OpenVPN server. Note that currently this will only work on Linux.



*The OpenVPN Dashboard for a user that can connect to one server, clicking "**Script**" downloads a script that when run connects the user to a VPN*

Administrator's Guide

As an administrator you will need to create an OpenVPN service and an OpenVPN dashboard before users can create and connect to OpenVPN servers.

Adding OpenVPN Services

An OpenVPN service is used to allocate ports for all OpenVPN Tenants (which have a one-to-one relationship with OpenVPN servers) and create instances in a slice on which OpenVPN servers will run. You must configure a service so that your users can create, modify, and connect to OpenVPN servers.

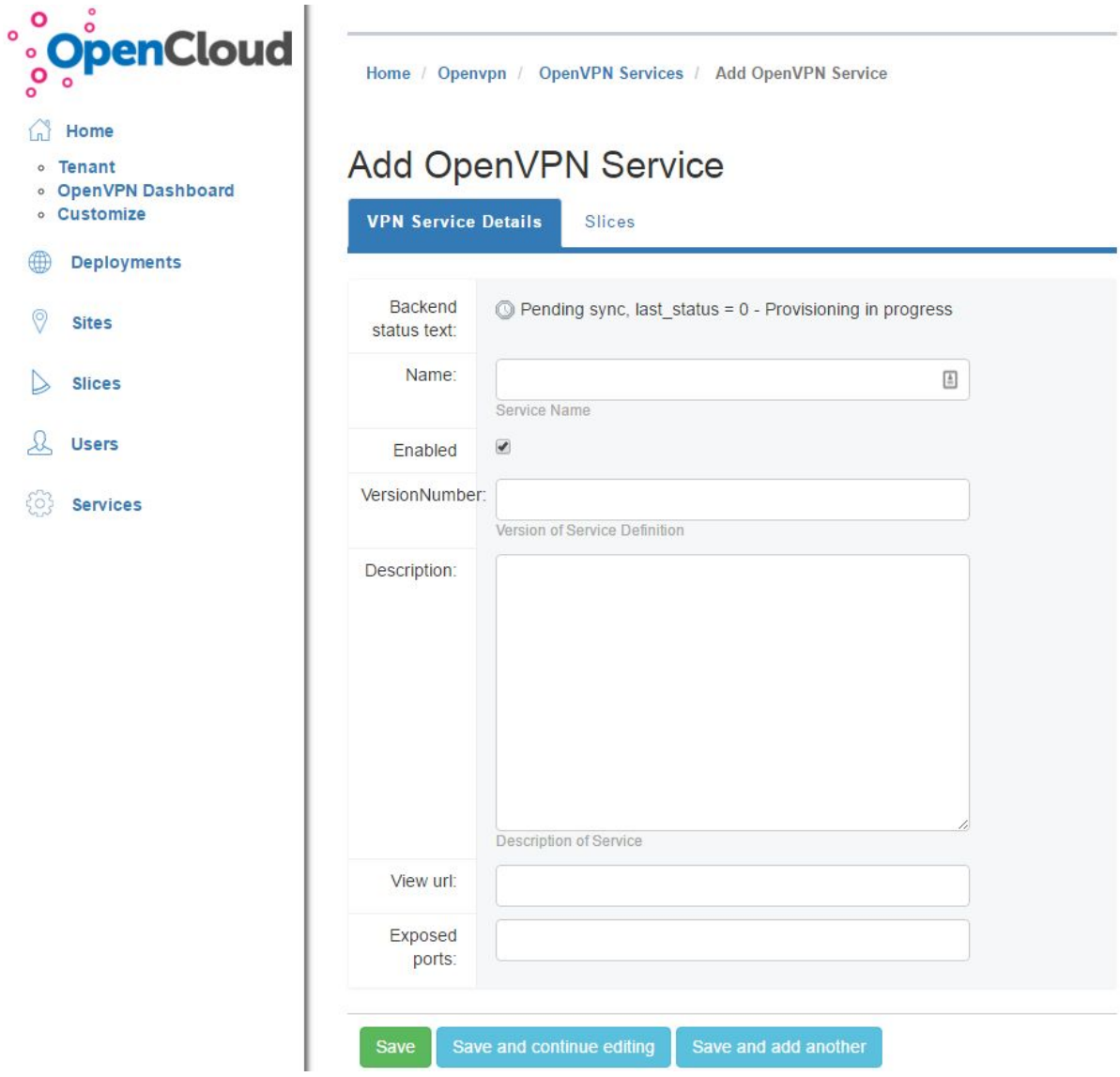
Step 1: Navigate to the service add form

Navigate to the admin form for a OpenVPN service through the XOS frontend, you can do this by navigating to https://[hostname]/admin/openvpn/openvpnservice/add/ where you will see the admin form.

Step 2: Create your new service instance

Fill out the form using your desired configuration. The options are described below. An asterisk denotes a field that is required.

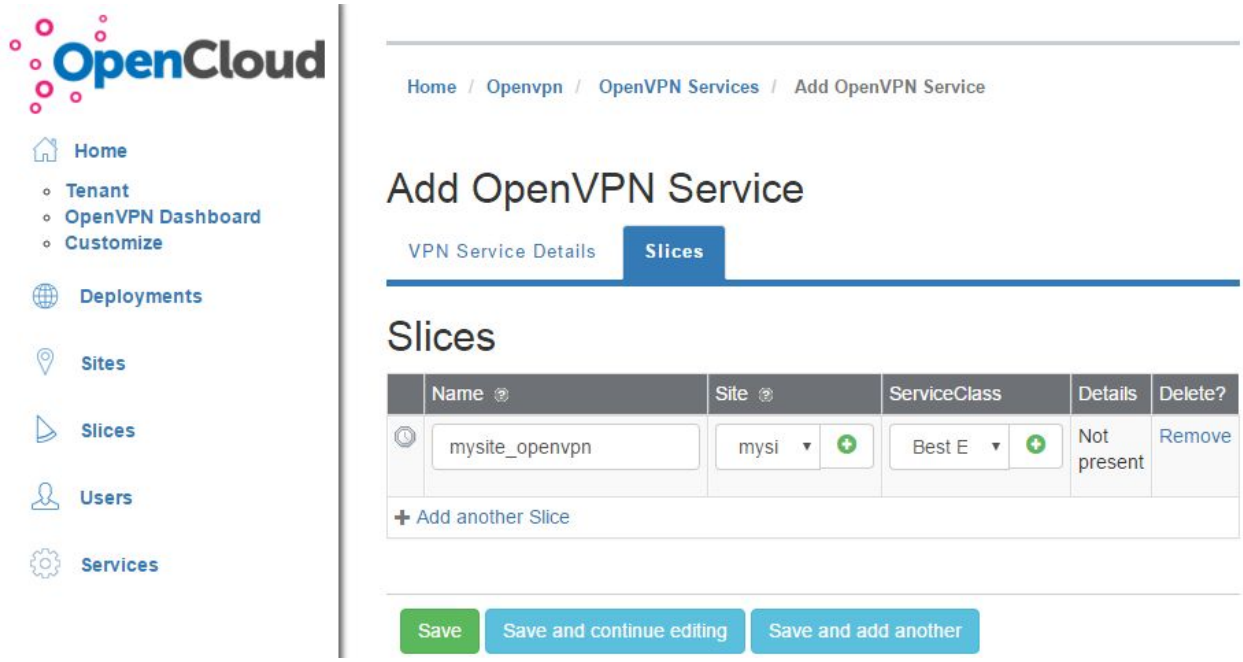
OpenVPN Service Details



The default Add OpenVPN Service form

- **Name***: the name of the service. This is used for identification and does not need to be unique.
- **Enabled**: whether or not this service can be used to create OpenVPN servers
- **Version Number***: The version number
- **Description**: a description to explain what this service instance is used for
- **View URL**: The URL to navigate to when a user accesses this services page in the left hand navigation area
- **Exposed ports***: a string representing the port numbers. Port numbers are specified with a protocol and a number or range separated by commas, for example TCP 4001:4012, UDP 5000. The port numbers specified here represent a pool that OpenVPNTenants will use to configure servers. One port number will be selected per server from this pool.

Slices



*Adding the slice "**mysite_openvpn**" to the new OpenVPN Service*

The slices tab is an inline for adding slices to the service, the **Add another Slice** link will allow you to create a new slice bound to this service. Slices hold virtual machines to run OpenVPN servers. The slice name must be unique.

When you are finished click **Save** and your service and associated slices will be created.

Adding an OpenVPN Dashboard

The OpenVPN Dashboard can be used by users and administrators to download scripts for the currently running VPNs. The script will create several files needed to connect to a VPN:

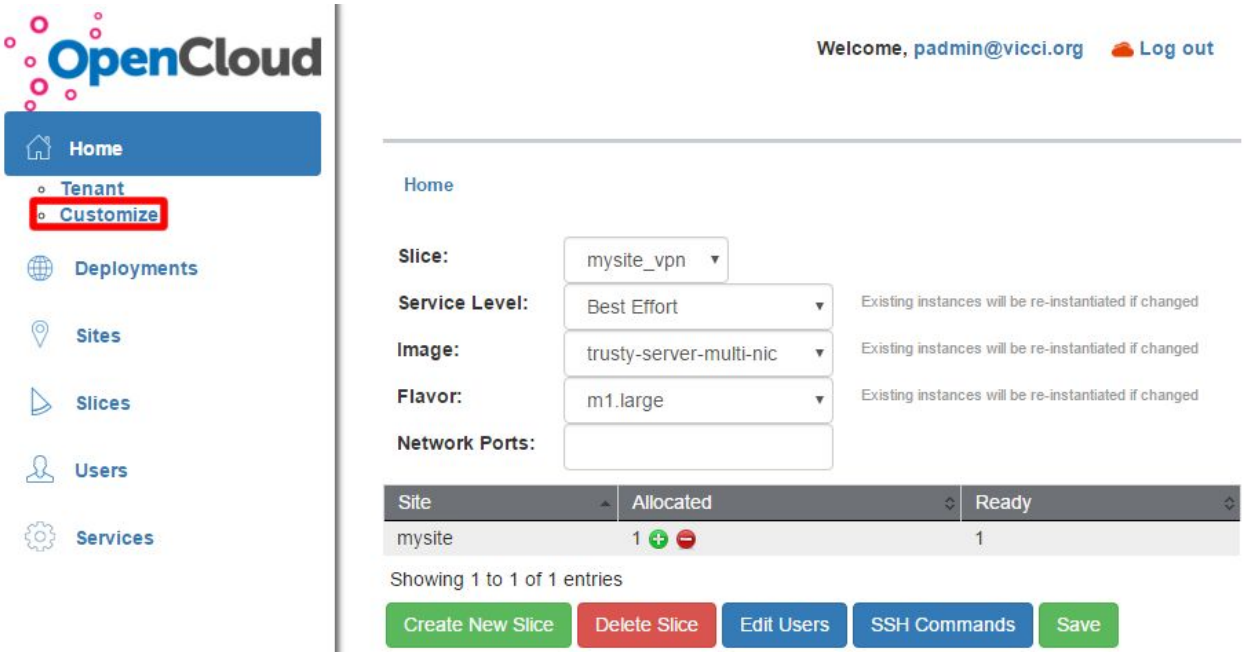
- A certificate for the Certificate Authority
- The certificate for the client
- The private key used by the client
- A client configuration files that reflects the options used when making a OpenVPNTenant

In addition to writing these files, the script will use apt-get to install OpenVPN and run OpenVPN with the client configuration. Note that no operations are run as root, however, OpenVPN may need root privileges to create the interface to the VPN. The script for the user cannot be updated when changes are made to the server, so if there are any changes that would require a new script (a change in server address, server subnet, or CAs) the clients will need to download a new script from the dashboard.

Add the dashboard for users

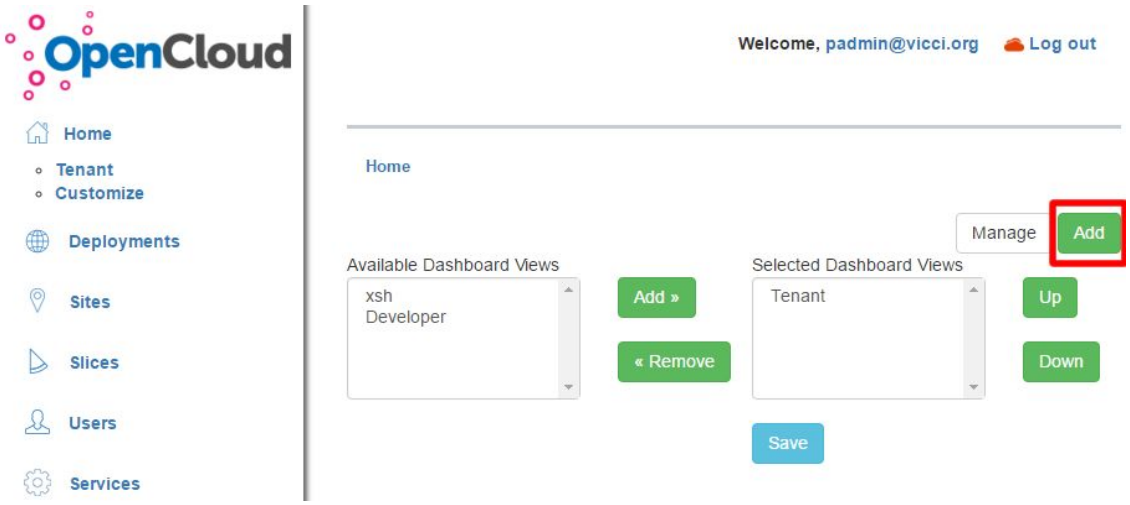
Adding the view follows the XOS development guide. You only have to do this once and it will be available for all users for every OpenVPN service.

1. In the left hand navigation click **Customize**



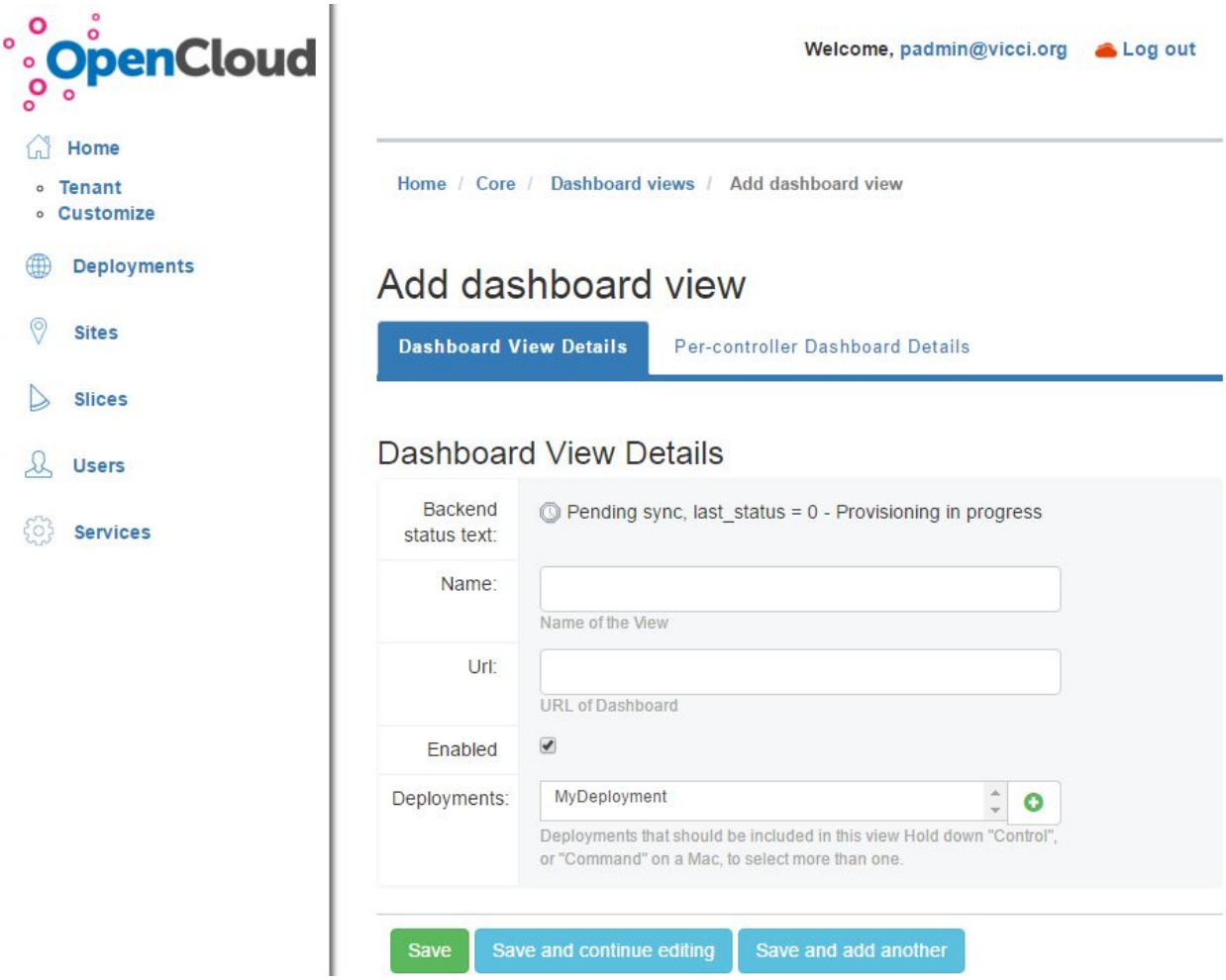
The Customize button

2. In the middle section click **Add**



The button to add a new Dashboard

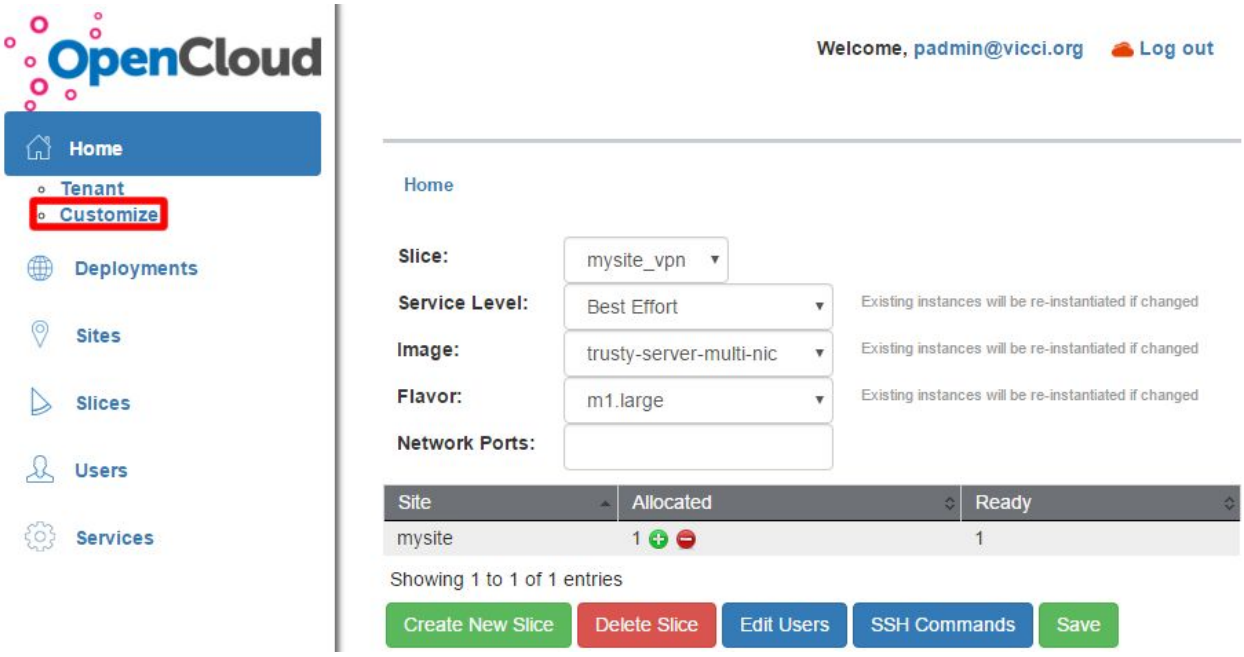
3. Fill out the form, the meaning of each field is described (all fields are required):



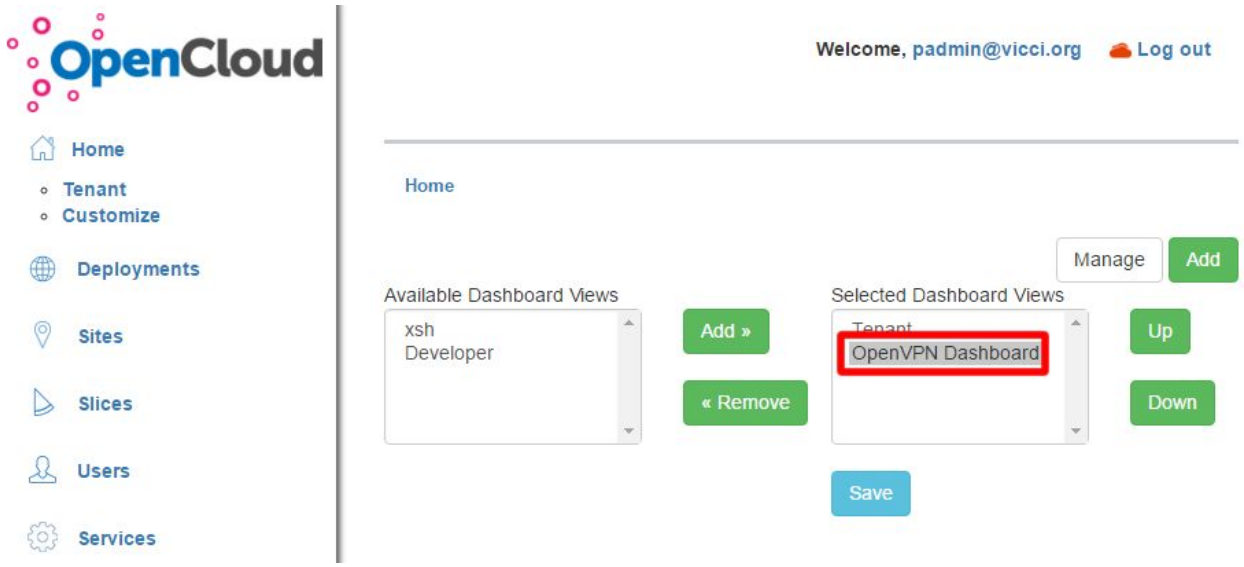
The default view to add a Dashboard

- **Name***: A descriptive name for the dashboard
- **URL***: the location of the dashboard the value for this field should be template:xosOpenVPNDashboard
- **Enabled***: If selected the dashboard is enabled
- **Deployment***: Select the deployment the dashboard is available from

4. When you are done click **Save**
5. In the left hand navigation click **Customize**

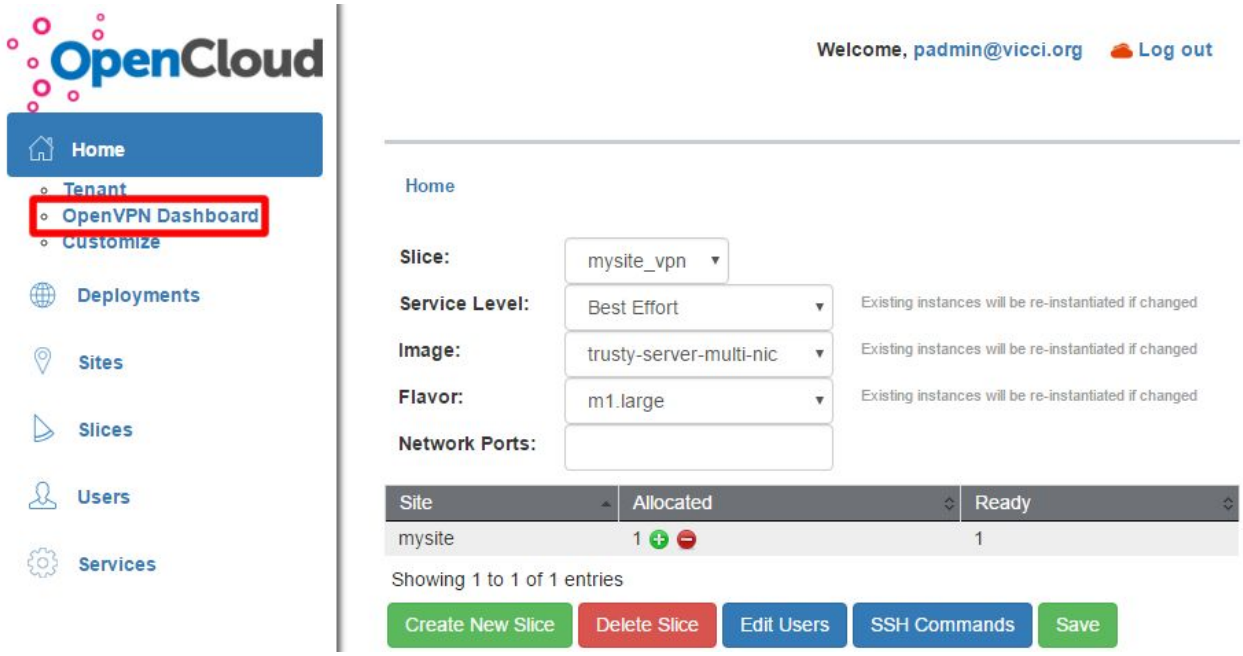


6. Add the OpenVPN dashboard with the name you selected then click **Save**



*Highlight the new Dashboard click **Add** and then **Save***

7. You can now access the dashboard from a link on the left hand side navigation area



The new OpenVPN Dashboard link

Developer's Guide

Behind the scenes the OpenVPNService and the OpenVPNTenants maintain a Public Key Infrastructure (PKI) which includes a Certificate Authority (CA), server files (a private RSA key, a public certificate containing the associated RSA public key and signed by the CA, and a Diffie-Hellman parameters used for key exchange), and client files (a private RSA key and a public certificate containing the associated RSA public key and signed by the CA). When we refer to the PKI we are referring to the collection of all of these files in the filesystem and their logical management. Each OpenVPNTenant has it's own PKI, and thus each OpenVPNTenant has it's own Certificate Authority.

OpenVPN Service

The OpenVPN service is implemented as a service in XOS. For the OpenVPN there are two important functions provided by the service.

Port allocation

The OpenVPN service is used by Tenants to acquire a port not used by any other OpenVPN tenant. This is important because if the OpenVPN servers are running on the same machine they each need their own unique port. There is a class method in OpenVPNService called `get_next_available_port` that takes in a protocol and returns a port number not already used by a OpenVPNTenant, or throws an error otherwise. The available ports are defined via

the **Exposed Ports** field of the Admin form. This field takes a string in the same format for specifying ports for a Network. Instead of providing this field explicitly, we could require that ports be specified as part of a Network on the slice for the service, but this design is more cumbersome. The ports field of a Network is stored as a list of strings of the form protocol:portRange, this makes it difficult to iterate over all available ports because the port range is not a single integer. Thus instead of doing this, we use a Map between a protocol name and a list of ports. This way when a Tenant needs to acquire a port it is simple to iterate over the list mapped to the specified protocol which is a more natural solution to the problem.

PKI Directory locations

The OpenVPN service has a method named `get_pki_dir` that takes in a OpenVPNTenant and returns a string with the location of the PKI for the tenant. This is the root of the PKI containing all of the files used for authentication. We put this method in the Service because all Tenants need a PKI directory and the service is a good common place to define the format of the directory names.

EasyRSA commands

The OpenVPN service defines a method named execute_easyrsa_command that takes in a string representing the PKI directory and a string representing the EasyRSA command to execute. Commands are executed using the batch option so no interaction is needed. All PKI interactions are done using EasyRSA so the OpenVPNService is a good place to hold this common functionality.

OpenVPN Tenants

OpenVPNTenants have a one-to-one correspondence with OpenVPN servers and VPNs, when you create a OpenVPNTenant a single server is created in an instance and that single server is the only server on a VPN. Most of the OpenVPNTenant is made on the typical way as every other Tenant class. The model for the OpenVPNTenant does nothing special besides setting fields that behave as described above. The synchronizer is composed of two SyncSteps, one for the OpenVPNTenant and one for the TenantPrivilege and performs many operations to maintain the PKI for the tenant. The OpenVPN server is maintained by the OpenVPNTenant SyncStep and manages these files:

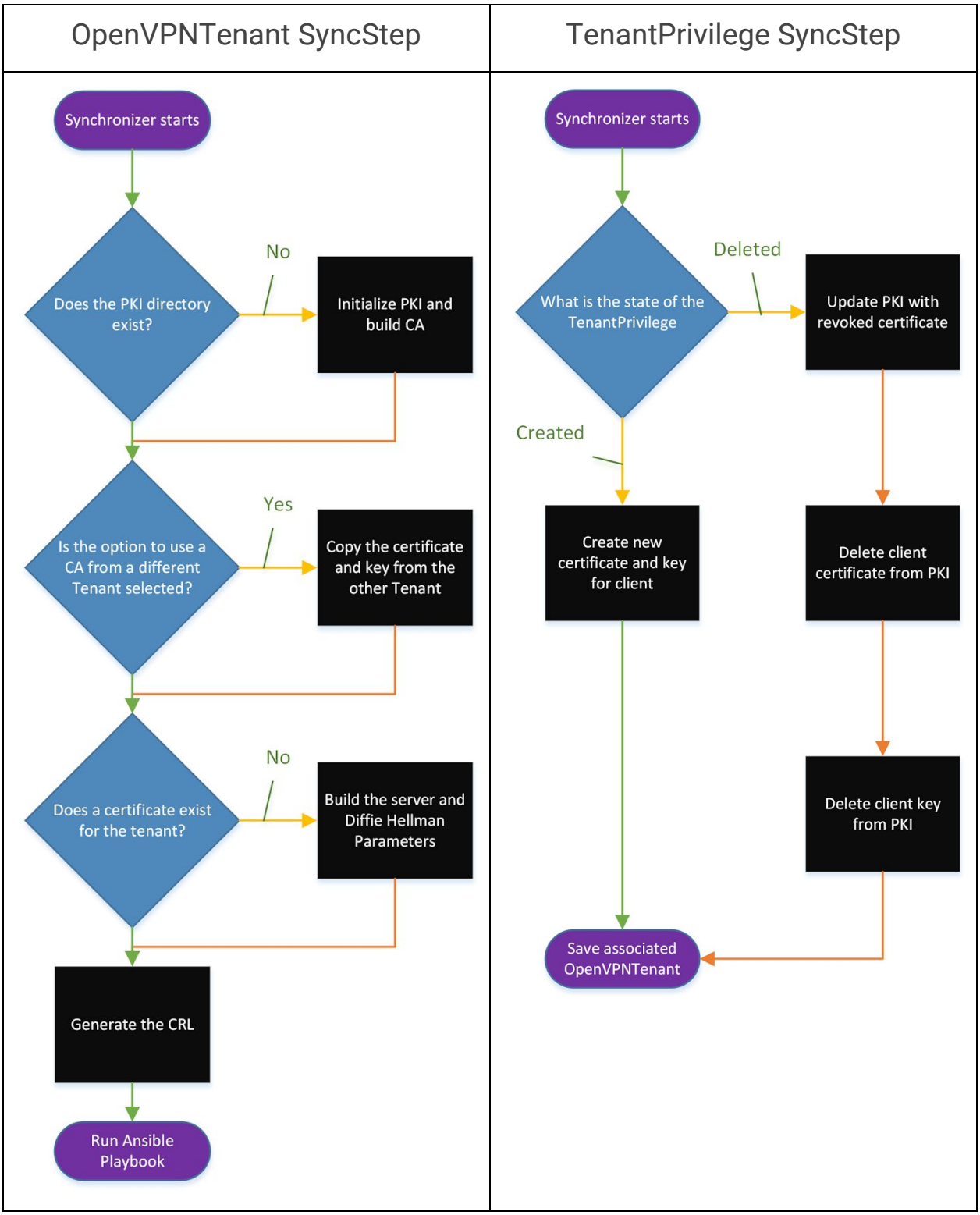
- **Certificate authority certificate (ca.crt)**: Used by the server to verify that connecting clients have certificates that are signed by the CA.
- **Server certificate (server-n.crt)**: A certificate signed by the CA sent to verify the server.

- **Server key (server-n.key)**: the private key used by the server for RSA.
- **Diffie-Hellman Parameters (dh.pem):** Diffie-Hellman primes used for key exchange.
- **Certificate Revocation List (crl.pem)**: Certificates that have been revoked.

The SyncStep performs various operations depending on the state of the data model. The TenantPrivilege SyncStep performs PKI operations when users are added to a OpenVPNTenant. The SyncStep creates or deletes the following files that are necessary for clients to connect:

- **Client certificate (<email>-n.crt)**: Used by the server to identify the client
- **Client key (<email>-n.key)**: Used by the client for RSA.

In addition to these files the clients have access to the ca.crt file but the TenantPrivilege SnycStep does not maintain the ca.crt file. When a client is created the TenantPrivilege SnycStep executes the necessary EasyRSA commands to generate these files. When a client is deleted the SnycStep issues a revoke command to update the PKI for the associated OpenVPNTenant to deny connections for the deleted client and to delete the files that were created for it. The behaviors of the two SnycSteps are given in the flowcharts below:


OpenVPNTenant SyncStep
Synchronizer starts
Does the PKI directory exist?
No
Initialize PKI and build CA
Is the option to use a CA from a different Tenant selected?
Yes
Copy the certificate and key from the other Tenant
Does a certificate exist for the tenant?
No
Build the server and Diffie Hellman Parameters
Generate the CRL
Run Ansible Playbook
TenantPrivilege SyncStep
Synchronizer starts
What is the state of the TenantPrivilege
Deleted
Update PKI with revoked certificate
Created
Create new certificate and key for client
Delete client certificate from PKI
Delete client key from PKI
Save associated OpenVPNTenant

Limitations

As a consequence of the OpenVPNTenant maintaining its own PKI, failover configurations are difficult to maintain. As mentioned in the User Guide, to create OpenVPNTenants that failover you need to use the **Use CA From** option to have one Tenant use the ca.crt and ca.key from another Tenant. You also need to verify that the users that are part of the failover configuration are users of both Tenants. Furthermore, it is not currently possible to have a tenant use the CA from another tenant and then switch back to using its own certificate. To do would require a larger history of certificates which is not manageable with the current design. In addition, by having the Tenants copy each other's CA private keys we are violating some security principles. A solution to this problem is available in the future development section.

OpenVPN Dashboard

The OpenVPN dashboard can be accessed as described in the User Guide, the code exists in the standard location as all other custom views. This dashboard displays a list of OpenVPNTenants to which the currently active user is associated. The script links on the dashboard can be used to download client scripts that write the CA certificate, and the client certificate, the client private key, and the client configuration. The CA will be the CA that the Tenant is using and the client and private key will be generated with the CA. The configuration will reflect some of the options chosen on the OpenVPNTenant form. In particular the client configuration will establish a persistent connection if the persistence option is selected and all of the failover servers selected when creating a Tenant will be in the configuration. Failover servers are used in order first with the Tenant and then every other server in order by ID. If the client cannot connect it will work down the list finding a server to connect to. The configuration for the client will also verify the server using TLS which prevents an attack where a client may pose as the server.

What I learned

This project serves as a capstone for my work in the CS program at the University of Arizona by involving a large amount of research of new technologies and through applications of skills taught in many of the core and elective courses in the program. I created two services in XOS, HelloWorldService and OpenVPNService, the research required is described below. At the end of this section is a description of how my coursework applied to developing these two services.

HelloWorldService

XOS is a relatively new project, and consequently when I started my thesis the documentation on creating new services was sparse. Thus my first task before creating the OpenVPN service

was to create a basic “Hello World” Service that would stand as an example of how to create new services in OpenVPN. Working with the other developers on the project I explored the several requirements for implementing a new Service in XOS. The full results of this research can be found in my original documentation for the service, but put simply I discovered that adding new services requires adding new models that represent the data and functionality for the service in the Django Data Model, and adding an Synchronizer that runs an Ansible Playbook to configure an associated instance (VM) to reflect the state of the Data Model for the service. Much work was put into understanding how to work with XOS, starting XOS itself on CloudLab (a project designed to allow people to experiment with the cloud), making the the changes to the code in a clean and standard way, and designing a Service completely considering both usability through the UI and functionality through the Synchronizer. Many new technologies had to be learned for this phase in the thesis, including Python, Docker, Ansible, and Django. By the end I wrote a comprehensive guide for adding a new Service to XOS that was the baseline for the current documentation that is updated as XOS evolves.

OpenVPNService

The OpenVPN service expanded on the work of my HelloWorldService, using it as a basis to create OpenVPN servers that clients could connect to. This part of the thesis expended on the research from the HelloWorldService, as I gained a better understanding of XOS and the technologies it used including Jinja2 (a Python templating language), Docker Compose (a scalable way to handle creating Docker containers), AngularJS (a Javascript based method for creating modern responsive JavaScript web interfaces), the Django REST API (an API for using REST from Django), XOS lib (a REST API for XOS), and most importantly OpenVPN (a program to create VPNs) and Public Key Infrastructure that it required. The PKI requirements for OpenVPN are quite complex and to understand them involved research of public key cryptography: understanding RSA, certificate signing, certificate authorities, and Diffie-Hellman key exchange.

How this thesis serves as a capstone

This project serves as a capstone for CS program by using many of the skills learned in the core courses in the department, while also giving experience with using real world skills.

Objected Oriented Programming and Design (CSC 335): XOS makes use of Object Oriented programming and requires thought into complex Object Oriented Design. The Django data model is a complex object hierarchy designed to be scalable. Adding new objects to the data model involves establishing an understanding of the existing system and designing new classes that most effectively make use of the existing hierarchy.

Learning new languages (CSC 127A, 252, 352): Working with XOS involves quickly learning new languages and technologies that represent different paradigms. Primarily this involved

learning Python which is an interpreted language and Angular JS a framework for a functional language.

Operating Systems (CSC 452): XOS itself is an Operating System and understanding it requires knowledge from that domain. This involves understanding how to interact with the filesystem and considerations of blocking vs nonblocking operations.

Computer Security (CSC 466): The PKI management of OpenVPN involves many topics in Computer Security including cryptography through RSA, key exchange through Diffie-Hellman, digital signatures and certificate management.

Database Design (CSC 460): Django is used for XOS to maintain data models in a database. Django interfaces with a Postgre database and thus operations using Django involved knowing how a database works and the best design for maintaining a database.

Software Engineering: Adding a service to XOS is similar to real work in the software engineering industry. Working on XOS involved contributing to a large software code base simultaneously with many other developers. This is exactly what a person does as a Software Engineer in industry. XOS is also an open source project accepting contributions from many individuals, academic institutions, and business many open source projects are like this and in industry contributing engineers often contribute to open source projects. Furthermore, developing the OpenVPN service for XOS involved thinking about good design principles such as maintainability and extensibility, which is often involved in industry engineering problems. Lastly, to understand the requirements for the OpenVPN service involved communicating with my thesis advisor and others to determine the functionality necessary for use. In the real world this is what software engineers do all the time when designing new systems.

Thus this thesis combines the core competencies of the CS program and also it serves are good experience for industry level work.

Future Directions

Design improvements

The design of the OpenVPN service has two distinct problems with its design, the management of the PKI and the storage of files.

PKI Management

PKI management should be done through a service. When a PKI Tenant is created it will generate a new CA that will be used to generate client and server credentials. PKI tenants can be configured to be shared or private, and clients can be added to the tenant using the TenantPrivilege. The PKI Tenant should have the option to be a one to one tenant or a one to

many tenant. With the first choice, a PKI tenant may only be associated with one Tenant, with the other option that PKI can be shared with multiple other Tenants. Each OpenVPNTenant can associate itself with a PKI Tenant, when the OpenVPNTenant is saved server credentials will be created using the associated PKI. With the design sharing PKIs between Tenants is easy as when the OpenVPNTenant is saved it can choose the PKI Tenant of other servers it wants to be in a failover configuration with. Moreover, the OpenVPNTenants will no longer need to manually copy the credentials of other Tenants.

File Storage

Under the current design the XOS frontend needs access to all of the files created by the OpenVPNTenant. These files include, the CA certificate and key, every server certificate, key, and diffie hellman parameters, and every client certificate and key. To make these files available a new container is created with a mounted volume that is then referenced by the containers of the XOS frontend and the OpenVPNTenant synchronizer. This is a very heavyweight solution because it requires that an entire container is created just to serve as a mounting point for files. A better design would be to use existing services in XOS, such as Syndicate, to handle the file storage to avoid needing to make container changes for the OpenVPN Service.

Additional Features

There are several features that could be added to the OpenVPN Service:

- Adding an entire slice of VMs as clients to a OpenVPNTenant.
- Use the OpenVPNTenant to replicate a single OpenVPN server on every VM in an associated slice.
- Adding a mechanism for choosing the best order of servers for failover based on performance metrics.
- Adding additional configuration options

Conclusion

The result of this thesis was the creation of a useful and extensible new service to XOS and OpenCloud that satisfies a need to create new VPNs by using OpenVPN. Now it is possible for users to create new VPN servers easily and control the users that are permitted to access the VPN. The groundwork has been laid for future work to extend the OpenVPN service to apply it to slices and services. The many configuration options supported allow users to design a VPN or many VPNs that best suit their needs. By leveraging only open source software supported by a dedicated community the OpenVPN and its underlying technology can be easily changed and adapted to new design requirements. Adding a VPN feature as a service is

ultimately the best way to implement this feature because it leverages the XOS framework for scalability and extensibility. Moreover, the OpenVPN service is part of a community of services that can interact with each other, so not only can end users make use of the OpenVPN service to create VPNs on demand, services in the future will be able to create and VPNs easily.