SETTING UP A VIRTUAL STUDIO WITH CONDUCTOR AND SHOTGUN

Jesse Lehrman | Senior Pipeline Developer, Conductor Technologies
Carlos Robles | Senior Devops Engineer, Conductor Technologies

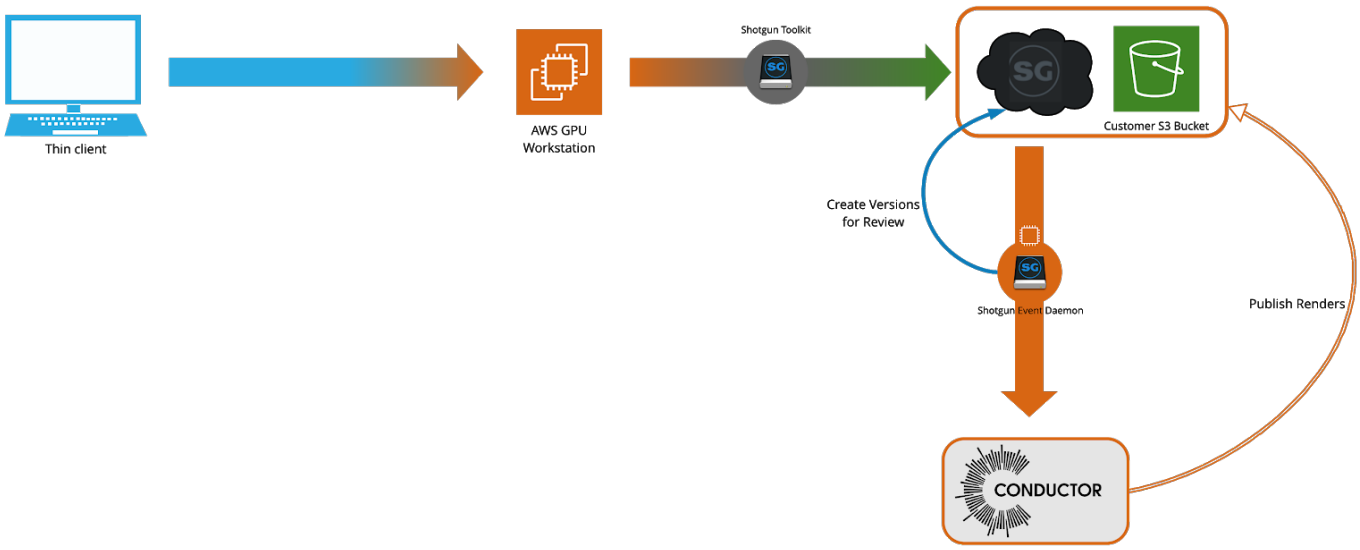
CONTENTS

INTRODUCTION

As the number of services offered by the major cloud providers expands, along with 3rd-party marketplace services, migrating a CG studio's pipeline into the cloud becomes more and more compelling. The mushrooming of services also increases the barrier of entry as knowledge of the cloud services landscape becomes crucial. As cloud services move away from the traditional bare-metal and virtualization techniques of on-premise infrastructure, it becomes increasingly difficult for those without a deep understanding of the cloud landscape to map a path to cloud services.

The goal of this whitepaper is to provide a working example of how a key piece of a CG studio's pipeline can be implemented entirely in the cloud.

The example used is a typical workflow of publishing and rendering a lighting scene (along with its precomp) to be reviewed via Shotgun, leveraging the Shotgun Toolkit:



Components

Before gettings started, it's important to have an overview of all the pieces of the puzzle:

Virtual Workstation

The virtual workstation removes the need for an artist to have physical access to expensive hardware. They effectively rent a machine that suits their requirements. The artist is only required to have access to a basic workstation (aka thin-client) and a sufficient internet connection.

Cloud Storage

Cloud storage will replace the shared file server that most studios have on-site.

Production Management System

There's not much change as all the popular systems (Shotgun, FTrack, etc...) are already available as cloud services

Pipeline Services

We're going to show how to take a service (the Shotgun event daemon) that traditionally runs on a local server (or VM) and port to run on a specific AWS service allowing us to take full advantage of the service's benefits.

Rendering

Despite marketing claims, no cloud provider provides infinite scalability. Conductor runs on top of multiple cloud providers and by leveraging multiple years of R&D and experience, Conductor's orchestration is able to scale to more render nodes than any other service.

Cloud Providers

The three major cloud providers; Google, Amazon and Microsoft offer comparable products. While most (if not all) of this whitepaper can be implemented in one of them, the paper uses Amazon's Web Services (AWS) for concrete examples.

Unfortunately, there is little common terminology across the three major providers, even though they largely offer the same set of services. It's left to the reader to map this guide onto one of the other cloud providers.

WORKSTATION

Workstation selection is straightforward. If using Teradici's Cloud Access Software (recommended) as the remote connection service, it's easiest to use their AMI which bundles the license and is set-up with NVidia drivers and CentOS 7.x. All that's needed is to install the DCC's of your choice. Unfortunately, there's currently no AMI's that include DCC packages. These would be the next steps to properly configure your workstation for use:

1. Install s3Fuse and auto-mount your cloud storage
2. Install the DCC's of your choice (and licenses)
3. Install the Shotgun Toolkit (SgTk)
4. Install the Conductor Client Tools

Once you have set-up the workstation for your needs, you can save the set-up as your own custom AMI so that it's easy to spin-up multiple instances.

STORAGE

Selecting the correct storage option is crucial. Not only must the storage be accessible from the various components but it must also be performant and suitable for all its uses. Examining all the various storage options is beyond the scope of this whitepaper. This whitepaper will concentrate on the features needed from a storage option and the performance it provides.

Due to how most DCC's and tools are designed, it's imperative that the storage option be presented as a local file system.

A cost effective way to get started, is to use object storage (S3 on AWS) and s3Fuse. s3Fuse will allow you to mount any object storage filesystem as a local file-system.

PRODUCTION MANAGEMENT SYSTEM

Since Shotgun is a hosted service, no effort is needed to make it cloud-friendly. In some cases you might need to analyze Shotgun's security settings so that it can be accessed via the virtual workstation and other AWS services.

PIPELINE

For this whitepaper, we'll use the Shotgun Toolkit (SgTk) as the toolset for asset management. We'll walk through the configuration changes that are needed to successfully connect it to Conductor so that renders will be submitted automatically.

We'll also deploy the Shotgun event daemon - which is responsible for automating many pipeline processes - onto AWS ECS (Elastic Container Storage). ECS provides the advantage of deploying Docker containers with ease and allowing services to be scaled as needed.

Shotgun Infrastructure

Configuration

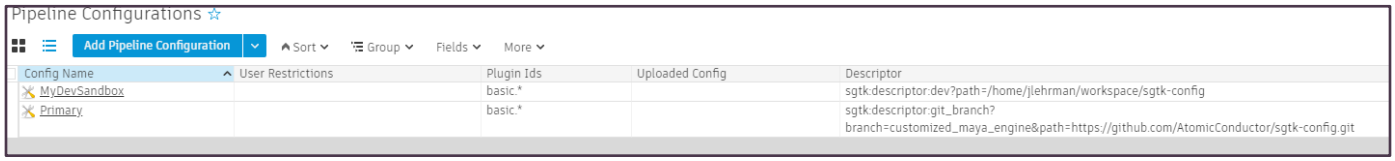
We'll rely on Shotgun's descriptors to configure the toolkit. For ease of use, we'll setup two configurations (both described as descriptors):

1. A local developer sandbox for testing and debugging
2. A distributed configuration for production (we'll be using `Primary`)

Since we'll be using git to store the configuration, the setup can be performed from any machine. Ideally it has the relevant DCC's installed - but it's not a requirement.

1. Follow the instructions to clone a configuration from your Primary (or other) config. Download the cloned config, create a new git repo and push it to the remote. You can also start from the config in one of the default repos.

Both configurations will be pulled from a git repository.



The first configuration, MyDevSandbox points to a drive location - which is a git clone of the Primary configuration. This allows for much quicker iterations than using a remote git repository when developing.

For more details on setting-up and configuring the SgTk, see the documentation.

Deploying the Conductor API

By adding hooks for the `tk-multi-publish2` app below, we'll need a mechanism to deploy the Conductor Python Client API.

It's best to deploy the Conductor API and its dependencies to shared storage so that it's accessible by all workstations.

Download the source of the of the latest release from:
https://github.com/AtomicConductor/conductor_client/releases/latest

Once the archive is extracted, use the requirements.txt to install the necessary dependencies:

```
pip install --target <target_dir> --requirement
<path_to_conductor_api>/requirements.txt
```

At the top of `conductor_collector.py` in the virtual studio guide repo, set CONDUCTOR_CLIENT_TOOLS_PATH to the root path to the Conductor API. Ensure that the two subsequent lines add the Conductor API and its python dependencies to sys.path

```python
import os
import sys

import maya.cmds as cmds
import sgtk

# Set this to the path for Conductor Client Tools here
CONDUCTOR_CLIENT_TOOLS_PATH = '/opt/conductor'

sys.path.append(CONDUCTOR_CLIENT_TOOLS_PATH)
sys.path.append(os.path.join(CONDUCTOR_CLIENT_TOOLS_PATH, "python", "lib", "python2.7", "site-packages")

import conductor.lib
import conductor.lib.maya_utils
```

Modifications

The key modification to the Shotgun Toolkit is adding a new collector to the `tk-multi-publish2` app. This new collector `conductor_collector.py` will extend how the publish app scrapes the Maya scene for dependencies by leveraging Conductor's dependency scanner. This ensures that all files required for a render have been added to the `PublishedFile` entity table. Out of the box, the `tk-multi-publish2` app has a very narrow dependency scanner in Maya.

`conductor_collector.py` inherits from the default `MayaSessionCollector`

A sample of how this can be done can be seen here.

To have `conductor_collector.py` be used by the `tk-multi-publish2` app in Maya, the sgtk-config will need to be updated:

`sgtk-config/env/includes/settings/tk-multi-publish2.yml`:

```yaml
# shot step
settings.tk-multi-publish2.maya.shot_step:
  collector: "{self}/collector.py:{engine}/tk-multi-publish2/basic/collector.py:{config}/conductor_collector.py"
  collector_settings:
      Work Template: maya_shot_work
  publish_plugins:
  - name: Publish to Shotgun
    hook: "{self}/publish_file.py"
    settings: {}
  - name: Upload for review
    hook: "{self}/upload_version.py"
    settings: {}
  - name: Begin file versioning
    hook: "{engine}/tk-multi-publish2/basic/start_version_control.py"
    settings: {}
```

Here, the shot step is being updated. The collector will also have to be added to all other steps that you intend to render from. All classes that `conductor_collector.py` inherits from precede it (separated with colons). This is a requirement for `sgtk.get_hook_baseclass()` to work properly.

NOTE

Having a complete list of dependencies stored in the `PublishedFile` entity table is essential to the process. When jobs are submitted to Conductor, the dependencies will be pulled from this table.

Shotgun Event Daemon

Deploying the Shotgun event daemon to a cloud service is a key component of this infrastructure. The Shotgun event daemon allows for automation. See below for an explanation of why the daemon was chosen over the new Webhooks.

Instead of deploying the daemon to an instance and running it in the traditional sense, we made use of AWS's Elastic Container Storage Fargate (ECS) cluster. This serverless, managed service removes the need to manage and monitor the computing resources required to run Docker containers. It also simplifies deployment. If the daemon is being overwhelmed, it's trivial to increase the resources so that it can crunch through the event queue quicker.

In all other respects, the Shotgun event daemon can be treated no differently than if it were running locally on-prem.

To build, deploy and run the Shotgun event daemon completely in the cloud, there are a few basic steps to automate the process:

1. Create a Dockerfile that builds a container that can run the daemon.
2. Using a CI/CD system (in our case, Codeship), write the steps to deploy the container to the AWS container registry.

At that point, ECS can be manually configured to pull the Shotgun event daemon container. However, we take the additional step of leveraging Hashicorp's Terraform to automate the creation of the infrastructure and services.

The Shotgun event daemon has a critical feature which enables it to resume from the last processed event even if it is unexpectedly shut down. This is accomplished by dumping details of the last processed event to a location on disk.

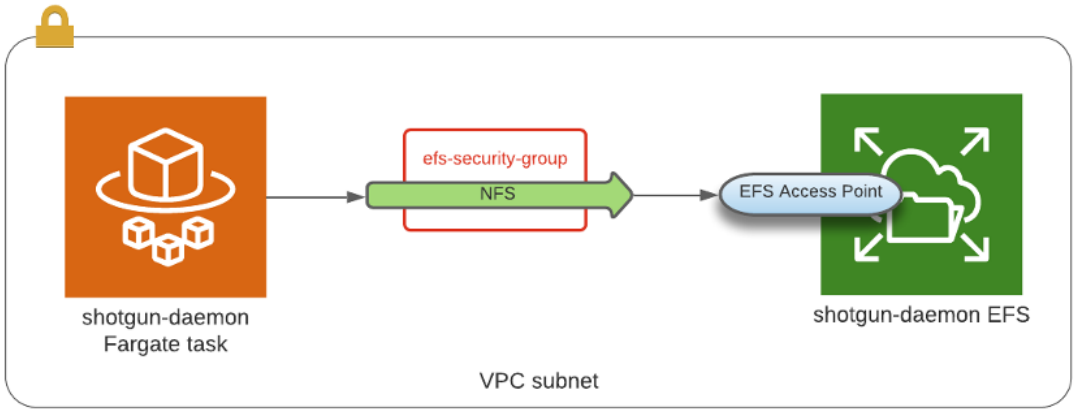
This presents a problem when using Fargate as there is no persistence between relaunches of the daemon - the data on disk will be lost. There are several approaches to mitigate this. We settled on attaching an EFS volume, which avoids altering the Shotgun event daemon code, provides a better place to deploy plugins and also gives us a place to persistently store the logs from the daemon.

Creating and attaching the EFS volume isn't complicated but there are multiple components involved. Familiarizing yourself with EFS, Security Groups and IAM Roles is strongly recommended before proceeding.

Attaching an EFS volume to the shotgun Daemon

An EFS volume needs to be created with an Access Point. Access Points allow other AWS services to access the EFS volume. That Access Point then needs a Security Group attached to it. A Security Group is essentially a firewall rule that defines what sources can access which ports. A source can be an IP or another Security Group. We're only concerned with inbound rules as outbound traffic tends to be left wide-open.

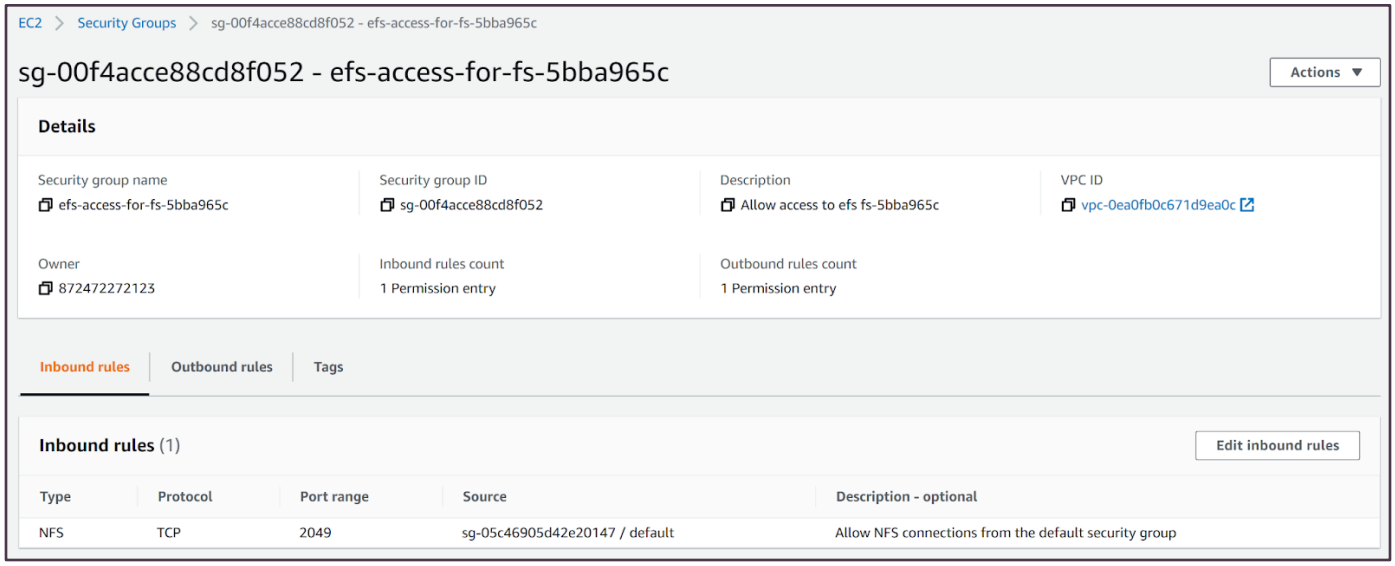
It's important to note that all this happens within the same Virtual Private Cloud (VPC), which is a mini-network within the AWS ecosystem.



NOTE

In the diagram above and the steps below, we refer to various entities by their names - however you'll have to use the entities ID's in the AWS console when entering values.

1. Create a new Security Group for the EFS volume that will allow incoming NFS connections from the default security group for the VPC.



2. Create the EFS Volume. The default options are fine. You'll want to ensure that it's in the same VPC as your ECS Cluster.
3. Create access points for the EFS Volume:
    a. Open up the details for the EFS and select the Network tab and then click on Manage.
    b. Ensure that the correct VPC is selected.
    c. Click on the Add mount target button. The Availability Zone and Subnet ID must be the same as the ECS Cluster. Select the Security Group you created in Step #1
    d. Click the Save button.

The EFS volume is now ready to be attached to the Fargate service. This is accomplished via the Terraform code.

Webhooks

For this integration we chose to use the Shotgun event daemon over Webhooks for a variety of reasons:

1. Currently, the industry has greater familiarity with the Shotgun event daemon, allowing for an easier transition of existing plugins
2. Migrating a traditional on-prem service to a managed cloud-service is a good learning exercise, allowing us to introduce new concepts.
3. Webhooks still require the connections to run somewhere (it could also be a server-less environment), so it isn't necessarily any faster to setup.
4. Webhooks are still in beta.

Ultimately running the Shotgun event daemon on ECS Fargate, is very similar to creating and managing our own Webhooks system. It may or may not be the right path for your studio. Read Shotgun's Webhooks documentation for a more in-depth comparison.

Pipeline Services

Now that all the infrastructure is set up for our integration, we can move on to actually getting it to do something. In this section we'll deploy a couple of plugins for the event daemon to automate the submission of renders to Conductor as well as make some changes to the Shotgun schema.

- `submit_maya_render_to_conductor.py`
- `create_version.py`
- `submit_nuke_template_to_conductor.py`

`submit_maya_render_to_conductor.py`

This is a plugin that will run whenever a Maya lighting scene is published (see the script for more details); It submits a job to Conductor to render (and publish the images) a published Maya scene.

It will query Shotgun for the scene files dependencies (stored in the `PublishedFile` table), copy the Maya scene and all its dependencies from the S3 storage bucket and submit them to Conductor. It will also generate all the meta-data needed to publish the render in the `PublishedFile` entity, store that meta-data in a `.json` file and upload it as part of the job so that it can be used to register the publish in the post-job command.

One novel enhancement of the Conductor job submitter script is the addition of a post-task and a post-job command. These are not built-in features of Conductor.

`create_version.py`

This is a generic plugin that will create a new Shotgun Version entity and associated media for review from an image sequence. This is triggered whenever a new image sequence is published. This allows for the automated creation of media that's reviewable directly in Shotgun.

The plugin will grab the frames from the S3 bucket, use `ffmpeg` to generate a compatible MP4, create the Version entity (attaching the MP4) and setting the status to "Pending Review".

`create_version.py` will automatically run once the lighting and comp renders are published.

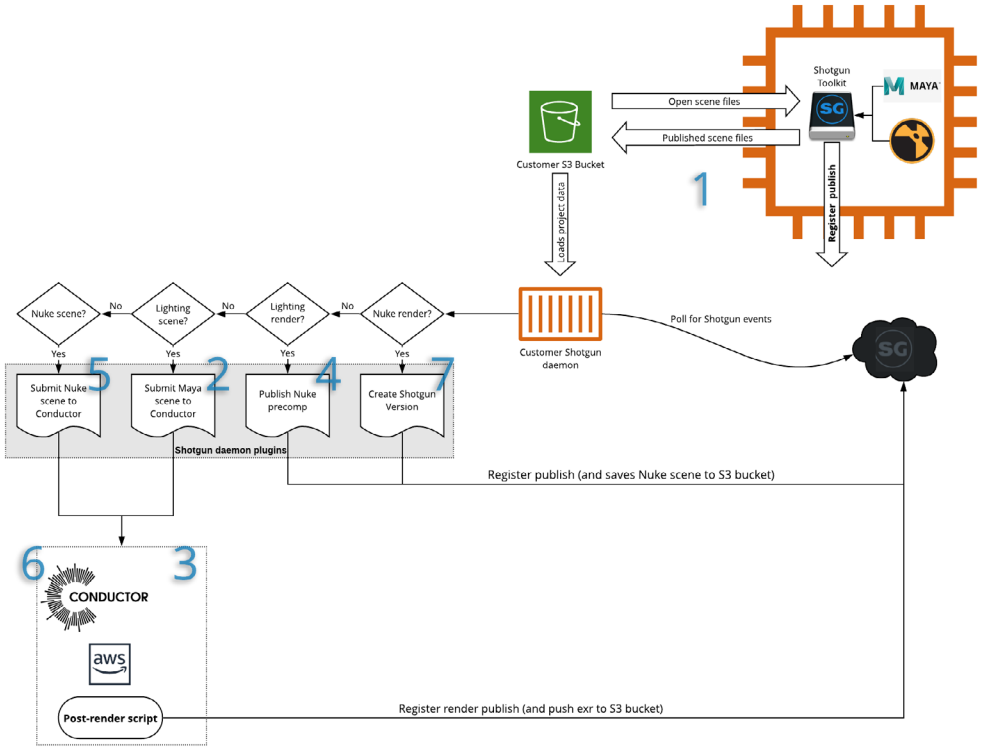
`submit_nuke_template_to_conductor.py`

This plugin is very similar to `submit_maya_render_to_conductor.py`. The main difference is that it's triggered by the publish of a 3D render. It uses that 3D render to generate a pre-comp with Nuke.

Like its sibling, it starts by pulling the frames from the S3 bucket. It also pulls a Nuke template file and all of its dependencies. The template is set up so that its read nodes reference command-line arguments passed to the Nuke command. This is a very useful feature of Nuke. See the official documentation and the sample template for more details. While the template is hard-coded in the plugin, querying Shotgun for a specific template would be more pragmatic.

Here, in addition to the post-job and task commands, we introduce a pre-task command. This ensures that the output path for the Nuke write node exists.

As in its sibling script, the post-task and post-job commands are used to transfer the frames and register the publish. These actions ultimately cause the `create_version.py` plugin to be executed.



A 'publish' equates to a PublishedFile entity being registered in Shotgun and files being properly saved to storage (in this case the S3 bucket)

Only the first step is manual. Everything else is automated.

1. Publish a Maya scene file with the SgTk (publish is registered with Shotgun and file is saved on S3 bucket).
2. Triggers the submit Maya scene to Conductor plugin on the Shotgun daemon. A job is submitted to Conductor with a post-render script.
3. Conductor renders the frame and pushes the result to the S3 bucket. If it's the last frame, it also registers a rendering publish in Shotgun.
4. The rendering publish triggers the Publish Nuke precomp on the Shotgun daemon. A Nuke file is modified to use render from step #3. The Nuke file is published.
5. The Nuke scene publish triggers the Submit Nuke scene to Conductor plugin on the Shotgun daemon.
6. Conductor renders the Nuke frames and pushes the result to the S3 bucket using a post-render script. If it's the last frame, it registers a publish.
7. Once a Nuke render is published, it triggers the Create Version plugin on the Shotgun daemon. This creates viewable media in Shotgun for review.

CONCLUSION

This guide has demonstrated how it's possible to move a piece of a studio's pipeline entirely in the cloud and how to remove on-prem infrastructure from the process. While not trivial, most of the concepts in this guide can be applied to an entire studio's pipeline, resulting in an efficient and well engineered virtual studio.

By leveraging existing cloud services such as Conductor and Shotgun and building knowledge of the managed services offered by the cloud providers, cutting the cord to on-prem infrastructure not only becomes an option with a realistic timeframe but also cost-efficient compared to a DIY solution.

For more information on this, or other related topics:

contact info@conductortech.com

or visit docs.conductortech.com

CONDUCTOR