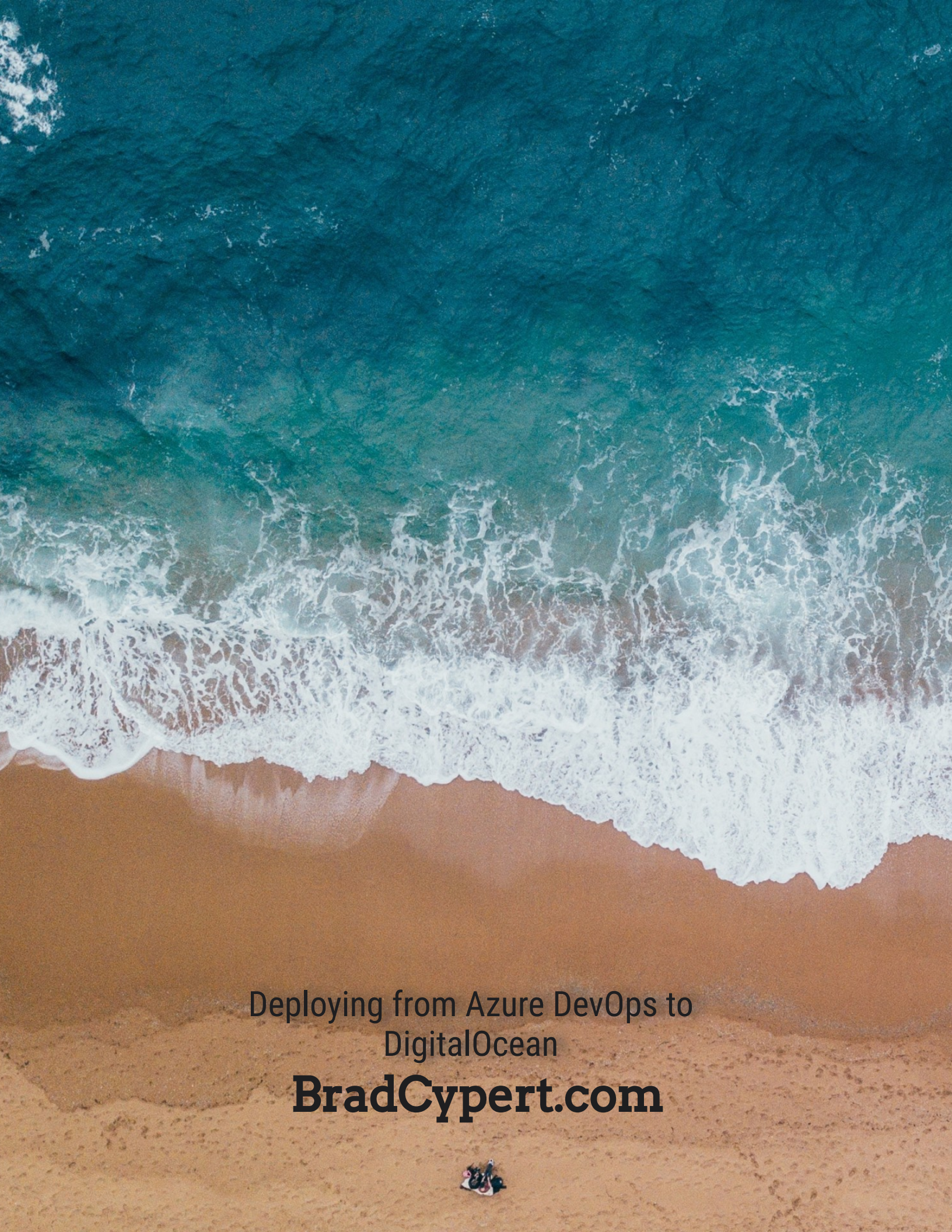


Deploying from Azure DevOps to DigitalOcean

BradCypert.com

Azure DevOps is one hell of a tool, but is it still an option if you don't want to deploy code to Azure? Well, yes, although it is a lot easier to deploy to Azure than anything else. This makes sense given how the two integrate and are owned by the same company. However, I'm going to show you how I deploy from Azure DevOps to DigitalOcean.

Deploying from Azure DevOps to DigitalOcean isn't as difficult as it may seem. In fact, with Azure DevOps' build pipelines and a few modules, you can deploy to most servers with ease. And best of all, Azure DevOps gives you free CI/CD minutes each month to take advantage of their build pipelines feature.

How we deploy Porios

My team at Pyre Studios deploys Porios via Azure DevOps. The frontend is built and deployed to Azure Blob Storage (more on that in another post), but the API is particularly interesting. We deploy our Golang API to a DigitalOcean droplet, but you can also use this guide to deploy to any server that supports SSH.

Setting up your DigitalOcean Droplet

If you haven't set up a server to deploy on, I strongly recommend considering DigitalOcean. They're a cost-effective way to get started and scale. Plus, their target audience is developers and it shows. They've written hundreds (maybe thousands?) of guides of accomplishing common tasks with their servers. Want to know how to setup Nginx with Let's Encrypt? They wrote that guide (and we followed it when setting up Nginx and Let's Encrypt for our API)!

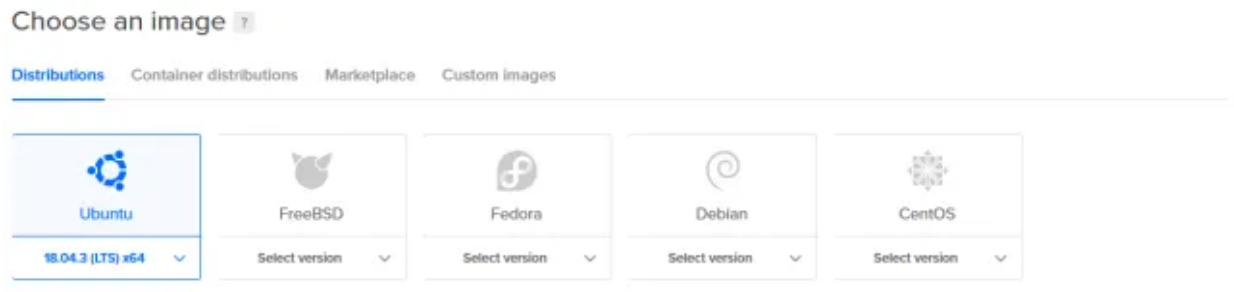
The first step with whomever you host your server through is creating an account. I'll let you handle that step on your own and then Ill show you how to setup your new server with DigitalOcean.

My First Droplet

Once you've setup your account (the rest of this guide will show the steps for DigitalOcean, but any VM provider should work similarly), you can create your first server. DigitalOcean calls these Droplets.

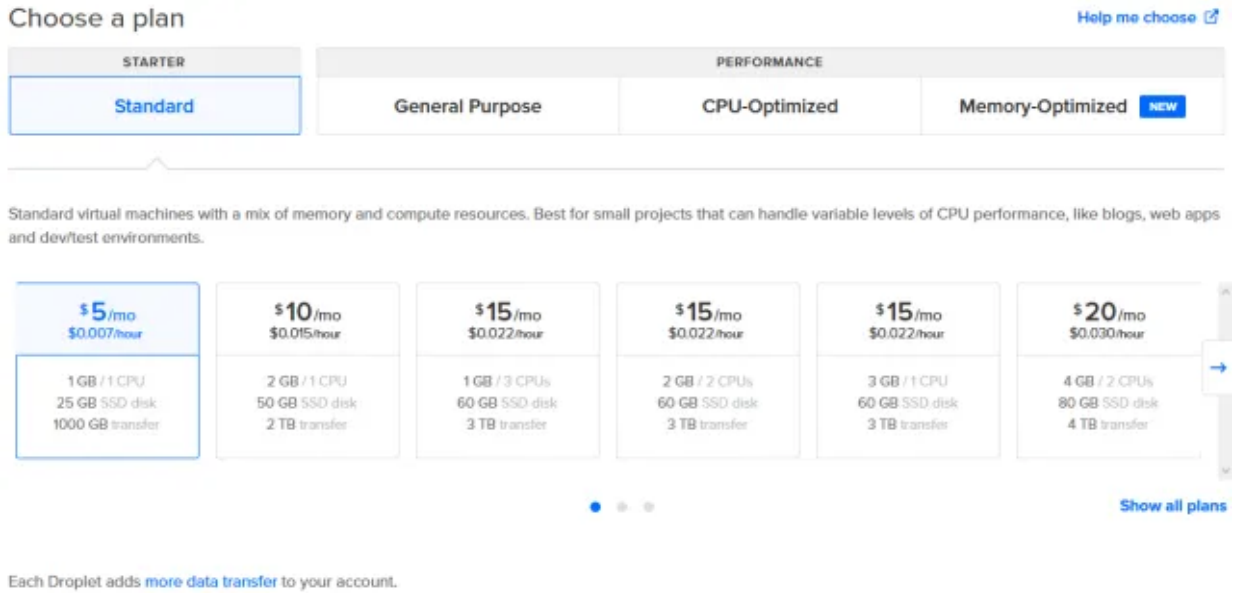
From the Logged In screen, search the top of the page and find something similar to the image below:

From there, select "Create" (the green button) and select "Droplets". You'll be presented with a rather long form. I'll show you what you probably want to change, but you can leave the default value for most of the items.



The first thing that DigitalOcean will ask you to do is choose an image. Feel free to choose what you're most comfortable with but the rest of this guide assumes that you've picked Ubuntu 18.04.X (it will likely work with other Ubuntu versions, though!).

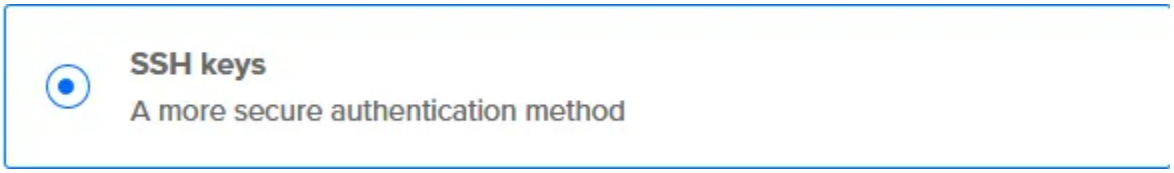
Next, You'll want to select your computing power. Don't worry, this isn't as overwhelming as trying to figure out what you need in AWS or estimating the cost of an Azure resource group. The cost and CPU/RAM is clearly defined and obvious — one of my favorite things about DigitalOcean.



Select Starter and $5/mo

As of the time of this post was originally written, DigitalOcean defaults you to a $40/month droplet. If you're just trying this out as an alternative, save yourself some money and try a $5/mo droplet. You can scale up easily afterwards if you need to. If you're committed to DigitalOcean, feel free to select whatever addresses your business/product's needs.

Once you've selected an image and a plan, you'll want to scroll down to the authentication section. I always recommend using SSH keys instead of the one-time password option and DigitalOcean explains how to set up your SSH keys underneath that option. In fact, the rest of this guide assumes that you're using SSH keys (as we'll need those to deploy from Azure DevOps)!



Finally, create that droplet and give it a moment to build! You'll be presented with an IP address. Copy that to your clipboard, as we'll use it in just a moment.

SSH Into Our Droplet

Now, we're going to create another SSH key-pair that'll use to communicate from Azure to our Droplet. Run the following command in your terminal to create a new SSH key in the PEM format:

```
ssh-keygen -m PEM -t rsa
```

Follow the steps provided and provide a passphrase to the key. I'd recommend giving it a different name, like "azure" instead of the default `id_rsa`. Now, we'll SSH into our DigitalOcean. You'll need your IP address and your generated username (probably `root`).

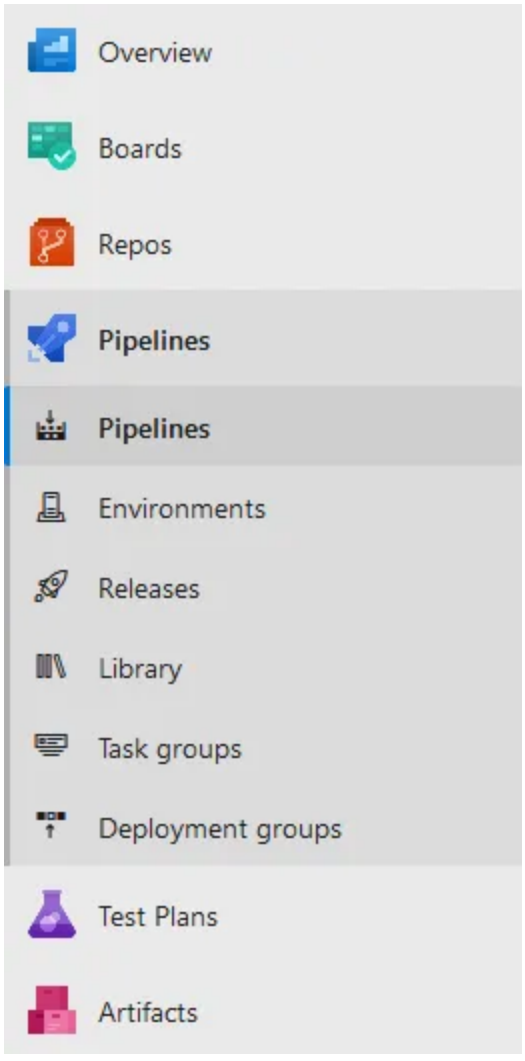
```
cat ~/.ssh/azure.pub | \
ssh root@1.1.1.1 "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys"
```

Run the above command, substituting your user and droplet IP address in place of `root@1.1.1.1`. If your user is brad and your droplet IP is 1.2.3.4, you'd instead write `brad@1.2.3.4`

This will copy over your newly generated public key and add it to the authorized keys section of your droplet. End your SSH session (`exit` in the terminal) and we can move on to setting up Azure!

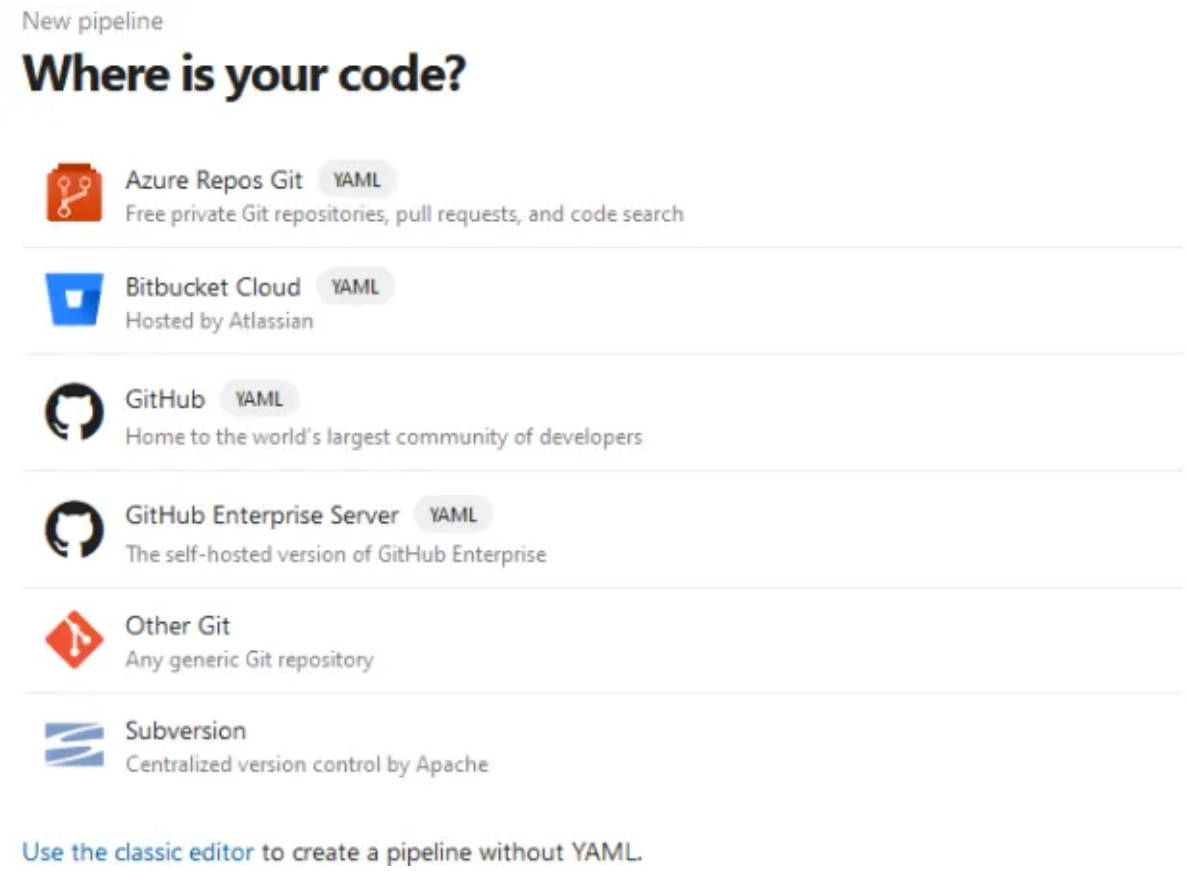
Setting up a remote resource in Azure

If you've made it to this point in the guide, Im going to assume that you're already using Azure DevOps. If you're not, I'll throw together another guide for that soon and update this post to link to it. For now though, we're going to proceed under the assumptions that you already have code you want to deploy in an Azure DevOps repository.



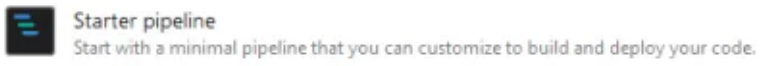
From your Azure Navigation menu, click on Pipelines and then click on Pipelines under the Pipelines menu item (they should really rename one of those).

This will take you to the pipelines screen. You'll want to create a new pipeline and select where your code lives (authenticate if necessary) and then select your repository from the list that Azure DevOps has found.



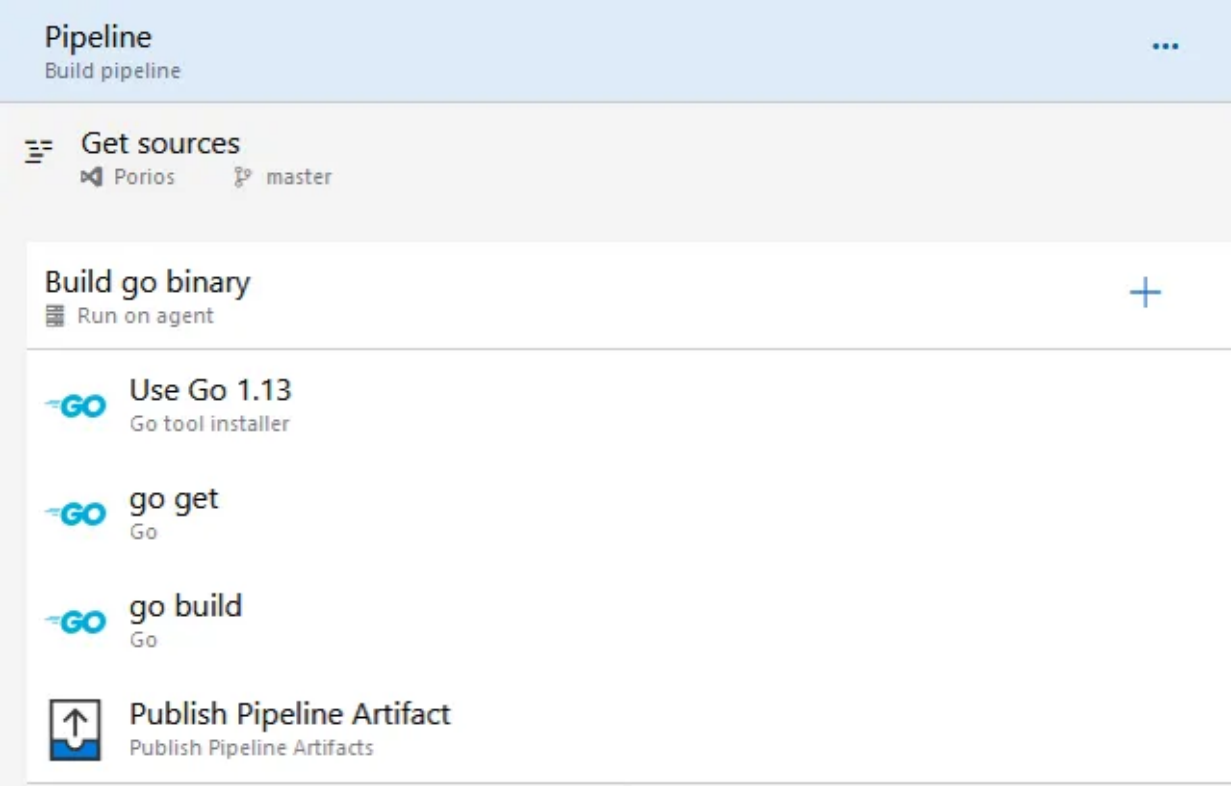
Select your repository provider and then select your repo

Next, it'll ask you if you want a pipeline starter. You can explore the pipelines that make sense for your application, but for now, let's just select the starter pipeline.



Once your pipeline has been created, we're going to go ahead and edit it. There are two things that our pipeline needs to accomplish. We need to build our project and then upload our project as an artifact to Azure DevOps. Later, we'll write our release pipeline that takes artifacts from Azure DevOps and pushes them our DigitalOcean droplet.

My pipeline for our Golang API looks like this:



We get sources for our project, install Go 1.13, fetch our dependencies with `go get`, build our app with `go build` (this produces a `main` file) and then we publish the main file as an artifact. The first three steps are specific to how we build our API, so if you're building a JAR or other binary, follow the steps for your build pipeline. There are modules for most languages, but you should be able to run shell scripts as necessary.

If you dont see a view like this, you likely need to add an job. Hovering over the top row ("Pipeline") should show you three dots. Clicking on that will allow you to add jobs. For now, you probably want an "Agent Job".

If you we investigate our Publish Pipeline Artifact, we'll see a view like this:

Our `go build` step in our pipeline creates a `main` file in our `server` directory. I'm telling Azure to pick that up as the artifact and name it `go-server` so we can easily find it when we create our release pipeline later.

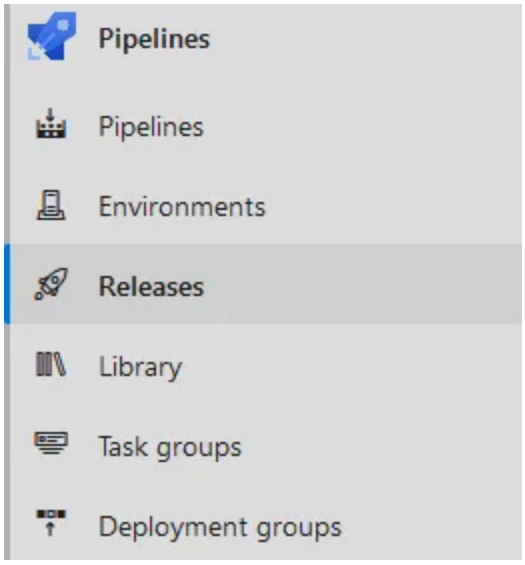
This should be all we need to get our CI pipeline setup. **Make sure you save your changes!** If you're a YAML person, the YAML for this agent is as follows:

```
pool:
  name: Azure Pipelines
steps:
- task: GoTool@0
  displayName: 'Use Go 1.13'
  inputs:
    version: 1.13
- task: Go@0
  displayName: 'go get'
  inputs:
    workingDirectory: server
- task: Go@0
  displayName: 'go build'
  inputs:
    command: build
    arguments: ' ./main.go'
    workingDirectory: server
- task: PublishPipelineArtifact@1
  displayName: 'Publish Pipeline Artifact'
  inputs:
    targetPath: server/main
    artifact: 'go-server'
```

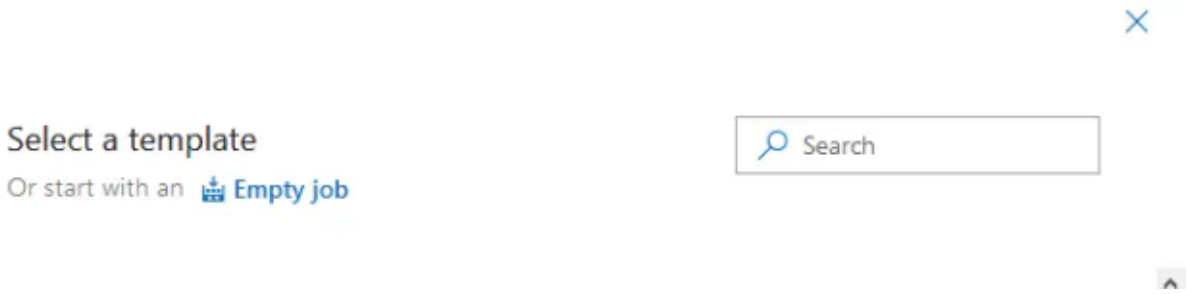
This YAML represents our pipeline, so yours might look slightly different.

Release Pipeline

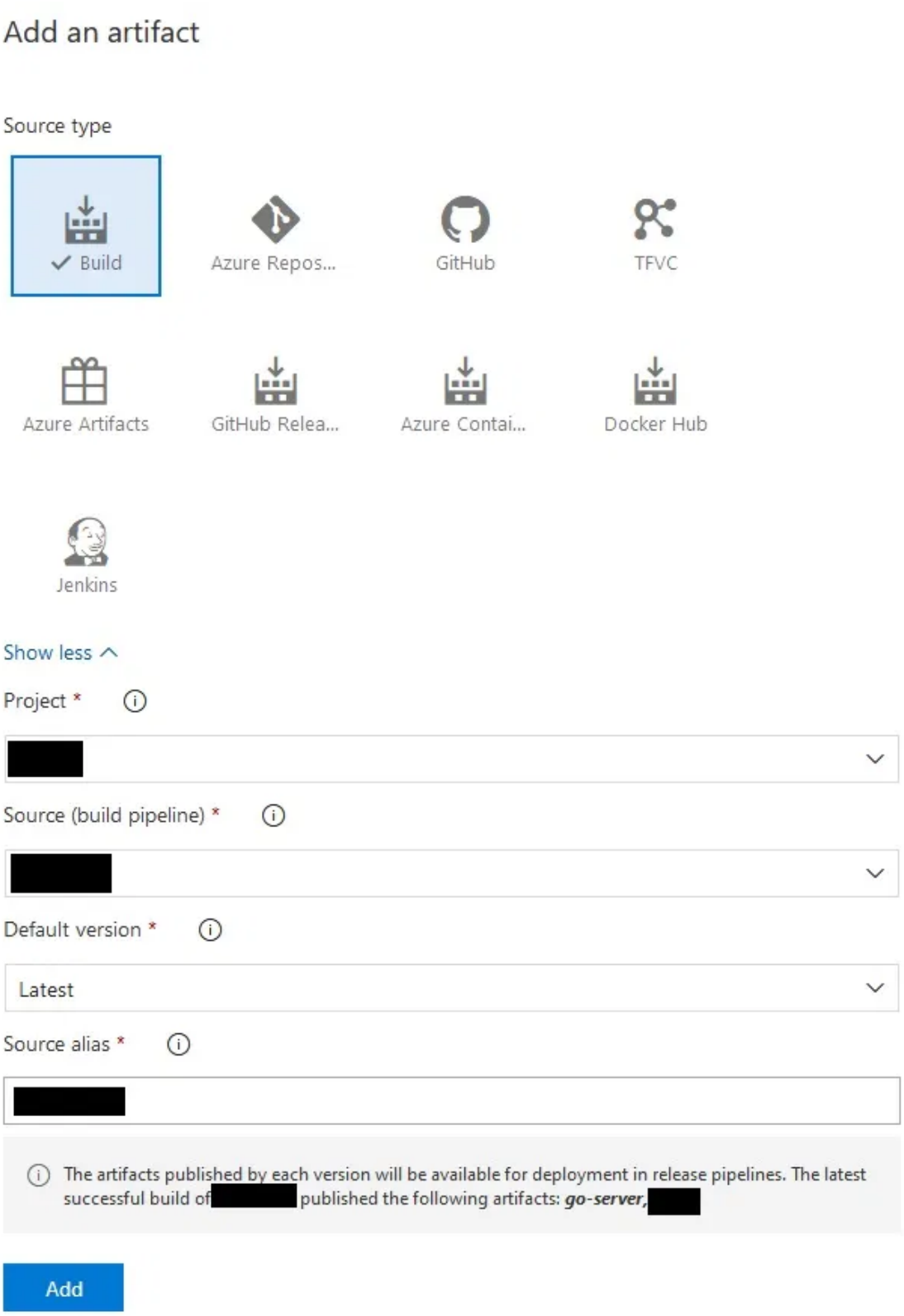
Finally, we can create our Release Pipeline. Our release pipeline will take our artifact from Azure DevOps and push it over SSH to our DigitalOcean droplet. I'll also give you an example of how you can run commands before/after the push.



Click on Releases and create a new release. You'll be prompted to select a template. Unfortunately, there's not a template for copying files outside of Azure, so we'll select "Empty Job" at the top of the "Choose Template" view.

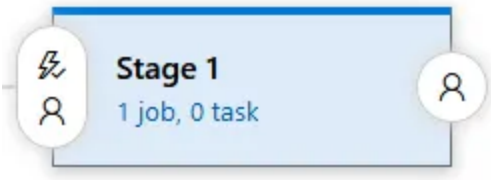


You should now see a small graph with two sections. First, click on Artifacts. You should then be presented with the artifact select screen.


Add an artifact
Source type
Build
Azure Repos...
GitHub
TFVC
Azure Artifacts
GitHub Relea...
Azure Contai...
Docker Hub
Jenkins
Show less
Project *
Source (build pipeline) *
Default version *
Latest
Source alias *
The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of [redacted] published the following artifacts: go-server,
Add

You'll need to make sure you select your project, source and should give the source an alias. You can name the alias whatever you want. Additionally, if everything is wired up correctly, you should see the artifact alias that we created in our pipeline step in the gray box above the "Add" button.

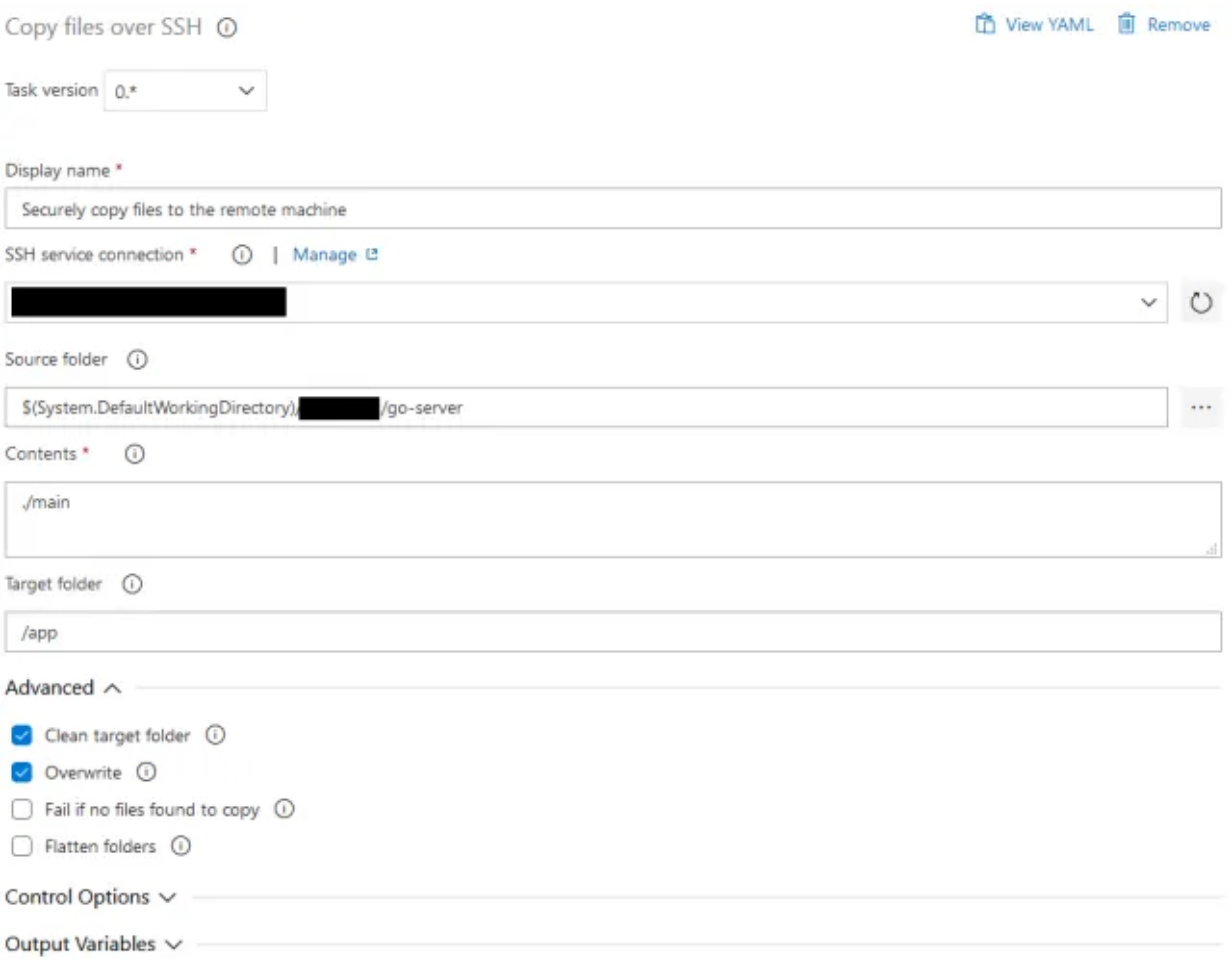
Add this artifact source and then click on "1 job, 0 task" in Stage 1 (or whatever you renamed this stage to).



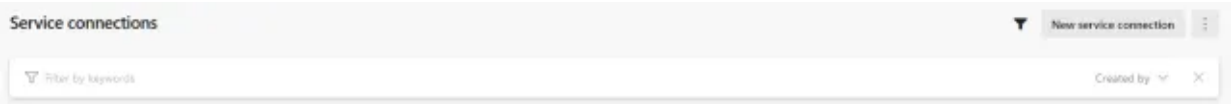
You should be presented with a familiar screen. You can rename your job and create tasks for it just like with a release. If you need an agent, click on the three dots next to the job name and select "Add an Agent Job."

The key task here is to copy files over SSH. Make sure you add a task for that module.

You'll want to setup that task to do something like the following:

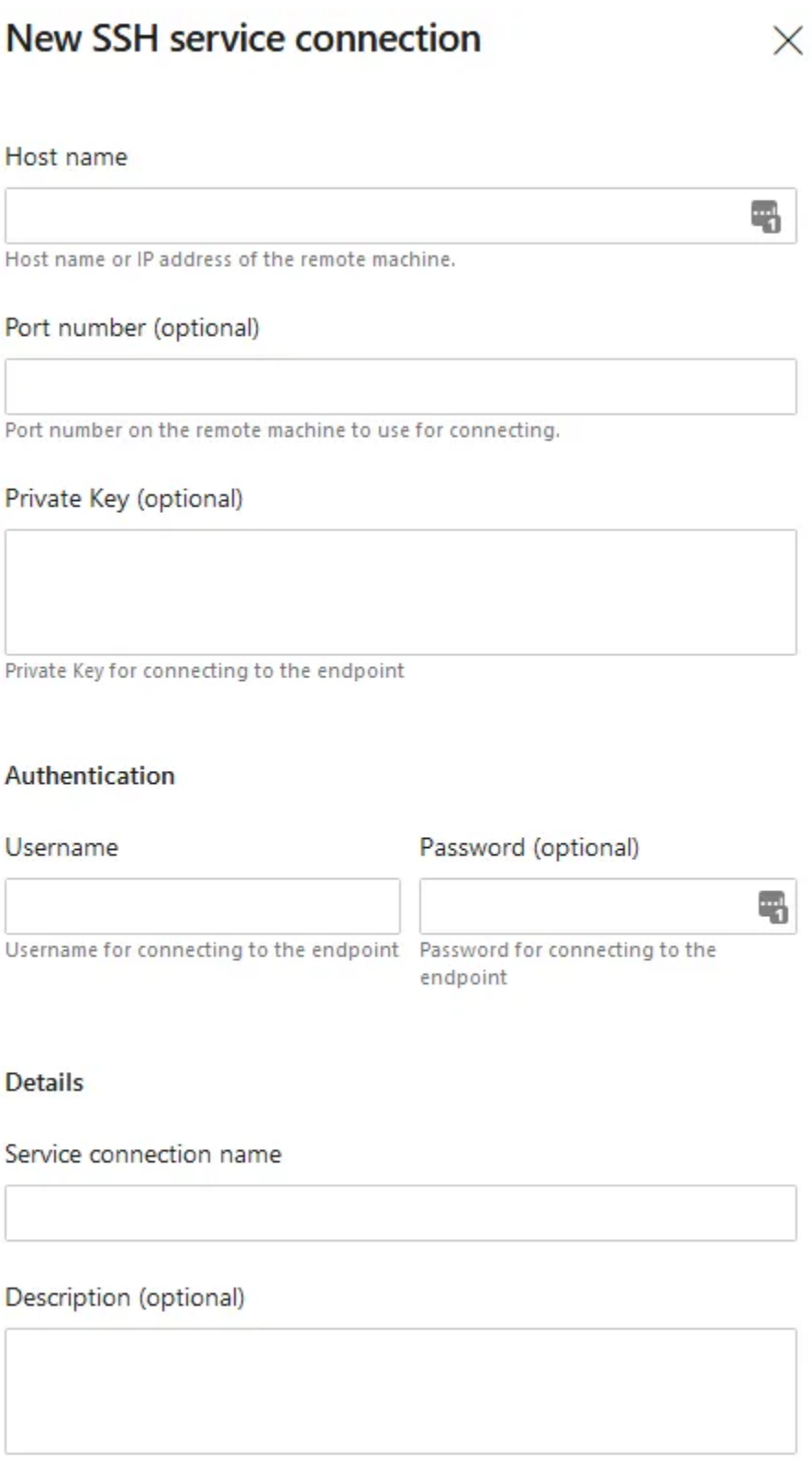


I've blacked out a few private values, but hopefully this is still fairly legible. Your SSH service connection isn't setup yet, so if you hit manage, you'll be taken to where you can set that up. Let's do that now.



Click on "New Service Connection" in the top right.

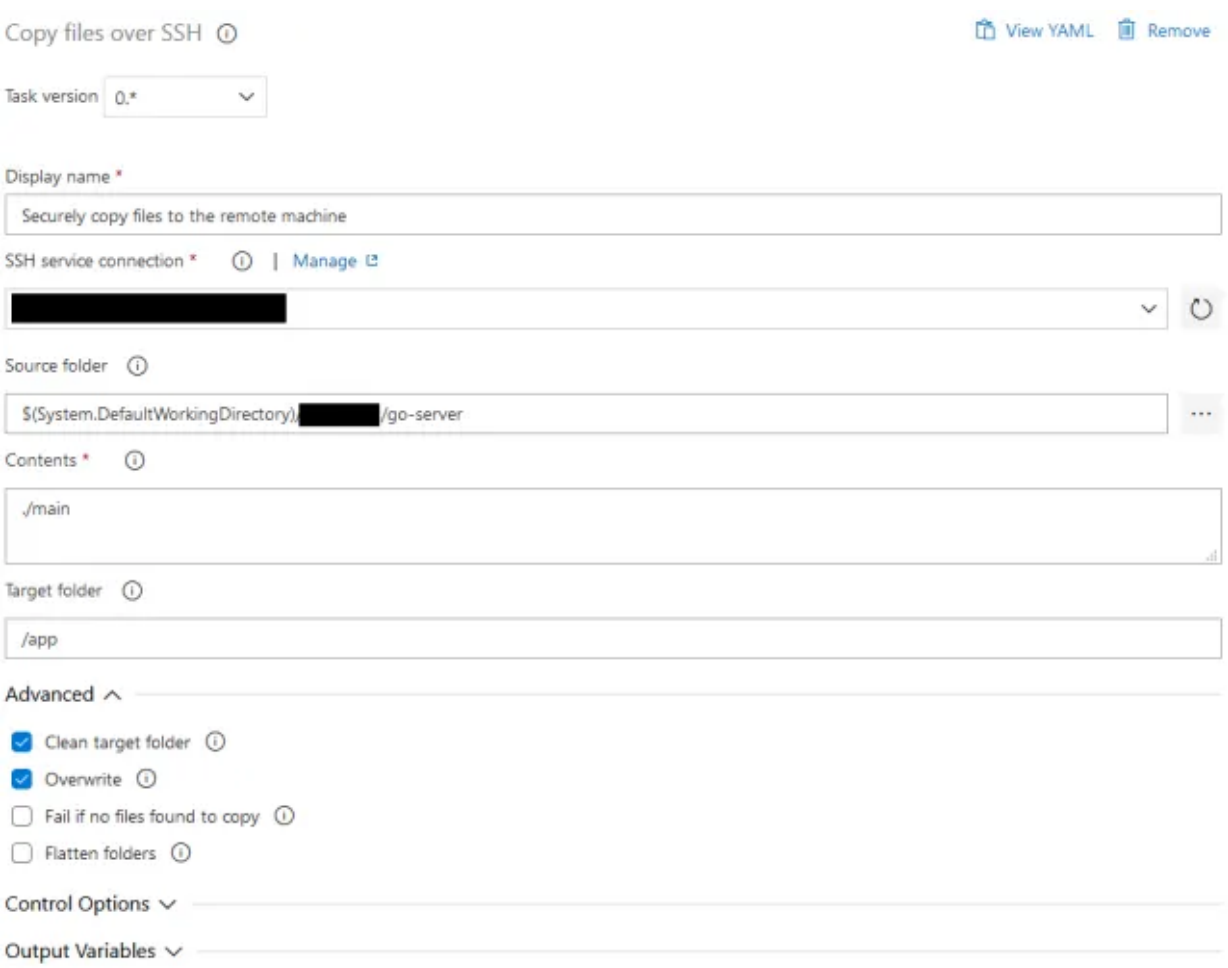
In the search box, type “SSH” and click on the SSH option below. We’ll now fill our our SSH service connection details.


New SSH service connection
Host name
Host name or IP address of the remote machine.
Port number (optional)
Port number on the remote machine to use for connecting.
Private Key (optional)
Private Key for connecting to the endpoint
Authentication
Username
Username for connecting to the endpoint
Password (optional)
Password for connecting to the endpoint
Details
Service connection name
Description (optional)

For hostname, provide the IP address of the DigitalOcean droplet. You can leave the port number empty, but you will need your private key. If you don't have a terminal open, fire one up and run `cat ~/.ssh/azure | pbcopy` to copy your `azure` key to your clipboard. If you named your Azure to DigitalOcean key something different, you'll want to copy that file instead.

Paste that value in the private key section of the form. For your username, you'll want whatever your username is on your DigitalOcean droplet is. If it's `root` put that in there. **Finally, if you added a passphrase for your SSH key, you'll need to add that passphrase in the "Password" field. This isn't obvious so I'll say that again. Add your SSH Key Passphrase to the Password field.**

Now we can go back to our release pipeline (its probably open in another tab) and select our new SSH connection. Here's what that should look like again.

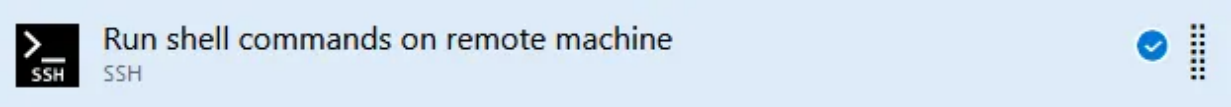


Its worth mentioning a few things that are specific to my pipeline. `go-server` (in the Source Folder) is the alias of my go artifact from our build pipeline. You'll want to use whatever you named yours instead. If you want to explore for that artifact, the button with the three dots to the right of this field should be helpful.

The contents is what in that alias directory you want to upload. In my case, the `main` binary. The target folder is where you want to dump the file(s) on your remote virtual machine. In our case, we dump ours in the app folder.

Now you should be able to save this and click “Create Release” at the top of the screen to create your first release. If you dont have artifacts, you’ll need to run your build pipeline (and validate that it is producing artifacts). If you can’t connect to the SSH server, make sure that your target folder exists and that you added the correct SSH key to Azure DevOps.

But how do you run the server? This is entirely up to you, but I run my server through systemctl. On my remote machine, I have a systemctl definition for the API that I deploy. Before I upload the files via SSH, I run an SSH command.

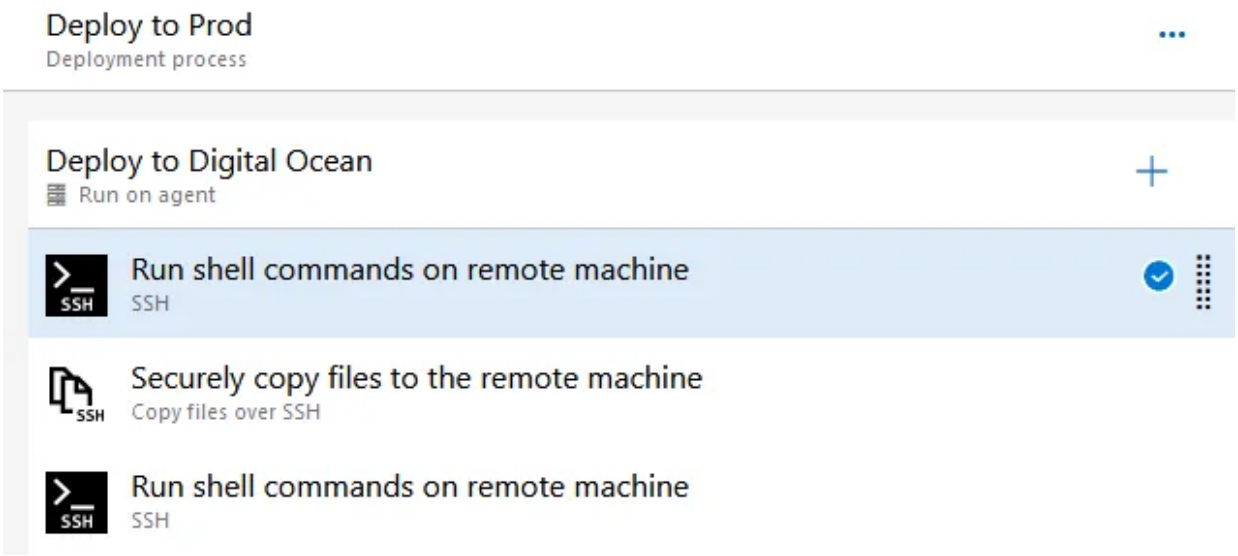



```
sudo systemctl stop my-api
```

And after I deploy, I run another SSH command.

```
sudo systemctl start my-api
```

This means that my task definition ultimately looks like this:



Conclusion

Hopefully you've found this be a helpful guide on your journey of deploying code from Azure DevOps to DigitalOcean. I was really hesitant about Azure DevOps (especially coming from Netlify for frontend work) but I've turned around and enjoy the build pipelines quite a bit. It's so much nicer than manually SCPing files from my laptop to a Droplet (we've all been there before)! Additionally, this lets anyone on the Pyre Studios team deploy our project at the click of a button.

Want to learn about other cost-effective tools for building a business? Check out that post here!