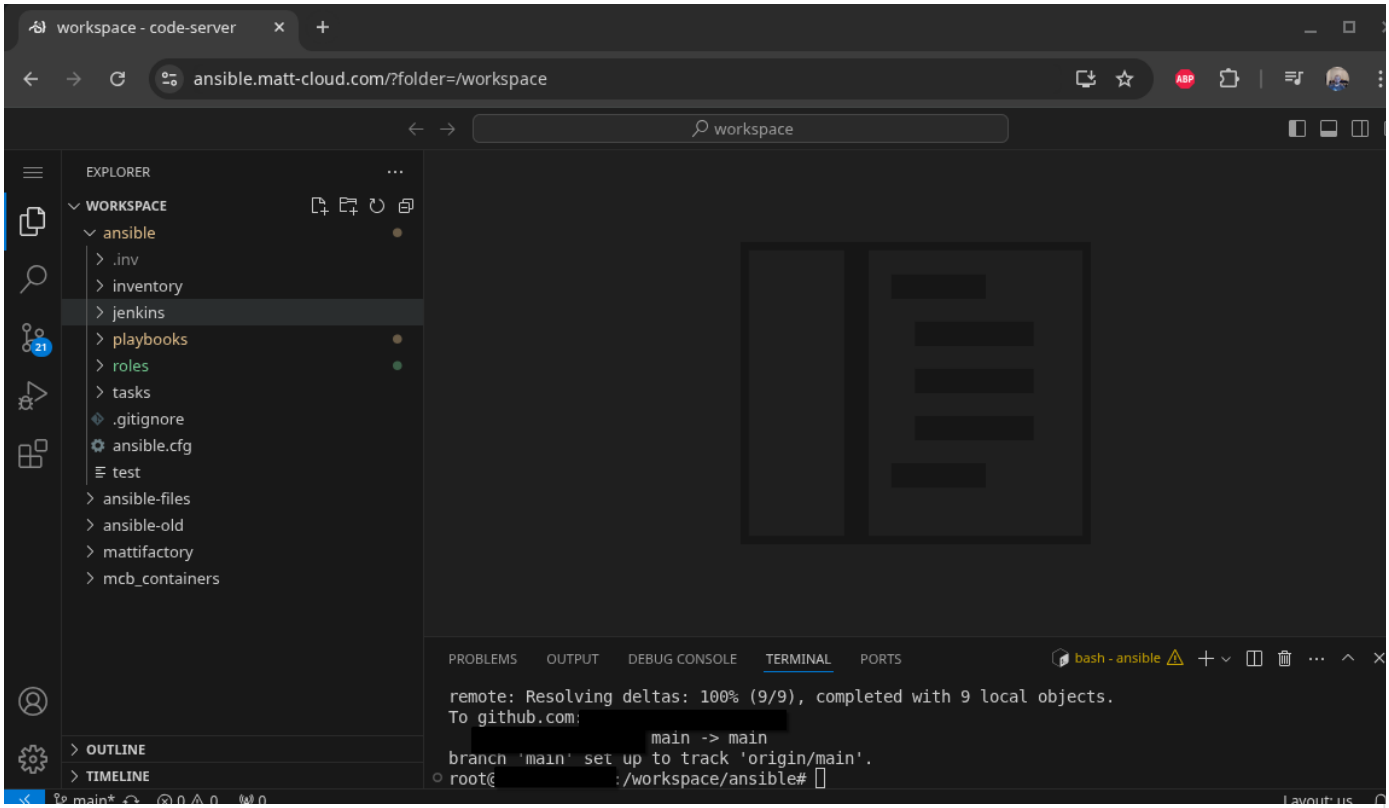
Jenkins and Ansible

I have since created a larger book covering this topic that is more thorough. It isn't very large, but it's more than this page covers. I have also copied this page over there.

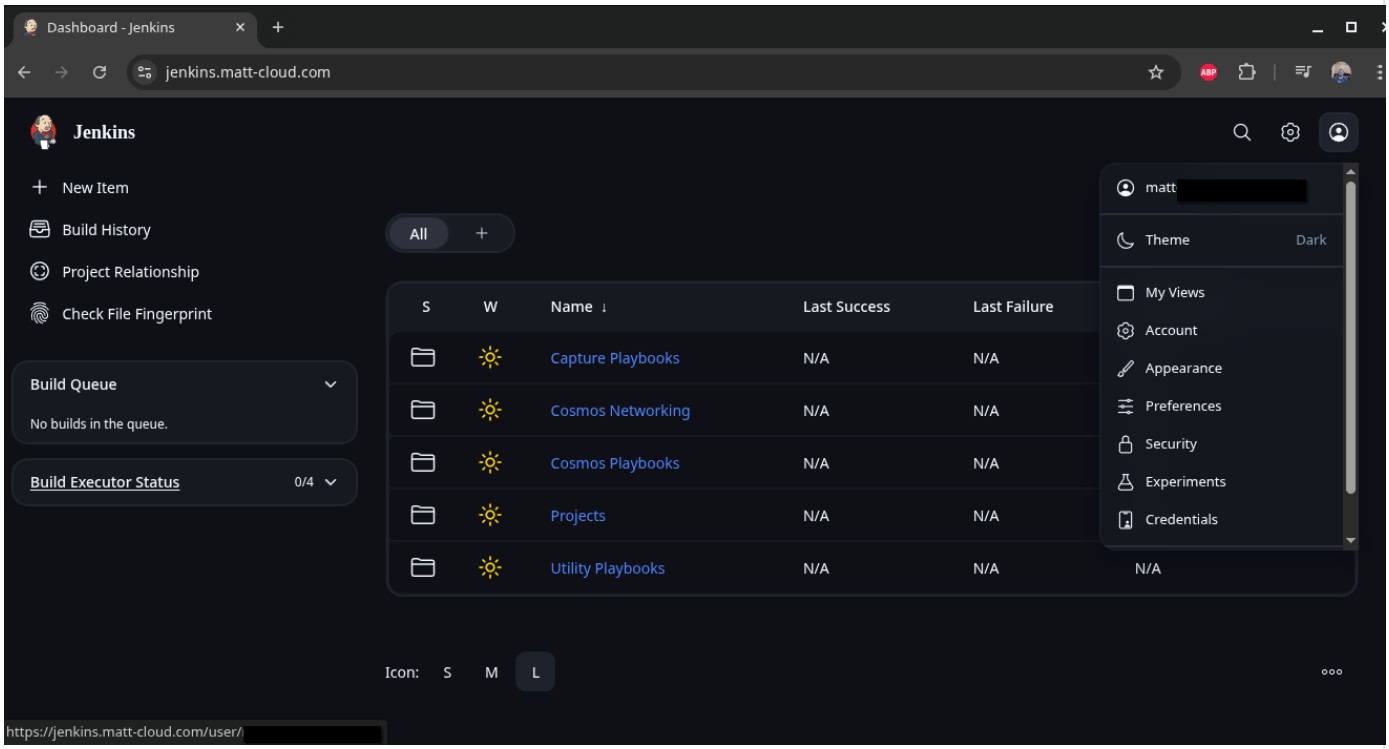
Alright so I figure I'll put down my thought on Ansible and Jenkins along with some of the source code. At the risk of being condescending I will start at the beginning. Jenkins is a web-based utility for running and logging unattended terminal commands. Ansible is a terminal based program for running lists of tasks on remote endpoints. The way I use these together is to use Jenkins to run Ansible. Ansible itself actually runs on the Jenkins host, and it connects to hosts based on inventory files. This requires having a working SSH key in place on your remote endpoints.

On my own Jenkins instance I use Github to sync my Pipelines with jenkinsfiles on my server, along with LDAP and OIDC for authentication which is way more complicated than a default Jenkins setup. You might have access; you can check your group membership at the Matt-Cloud Group page. Along with that, I use an open-source project to host all my pipelines and playbooks in a website, which itself is hosted behind my SSO. This is not something anyone but me has access to due to how insecure the access is. It is full write access to all files without any knowledge of the connected user. It works by effectively hosting files that live on a Linux system in the browser. I have that running in a docker container, with the paths of the ansible playbooks and jenkins pipelines in it so they can all be viewed and edited in the browser. I also have my code-server container customized so I can sync my github from the built-in terminal in the browser. Github lets me easily sync the Jenkinsfiles between the filesystem and Jenkins itself. Jenkins itself is configured in the browser, so the Docker settings for Jenkins are only a dozen lines or so of mostly volumes. Here are some screenshots of my stuff

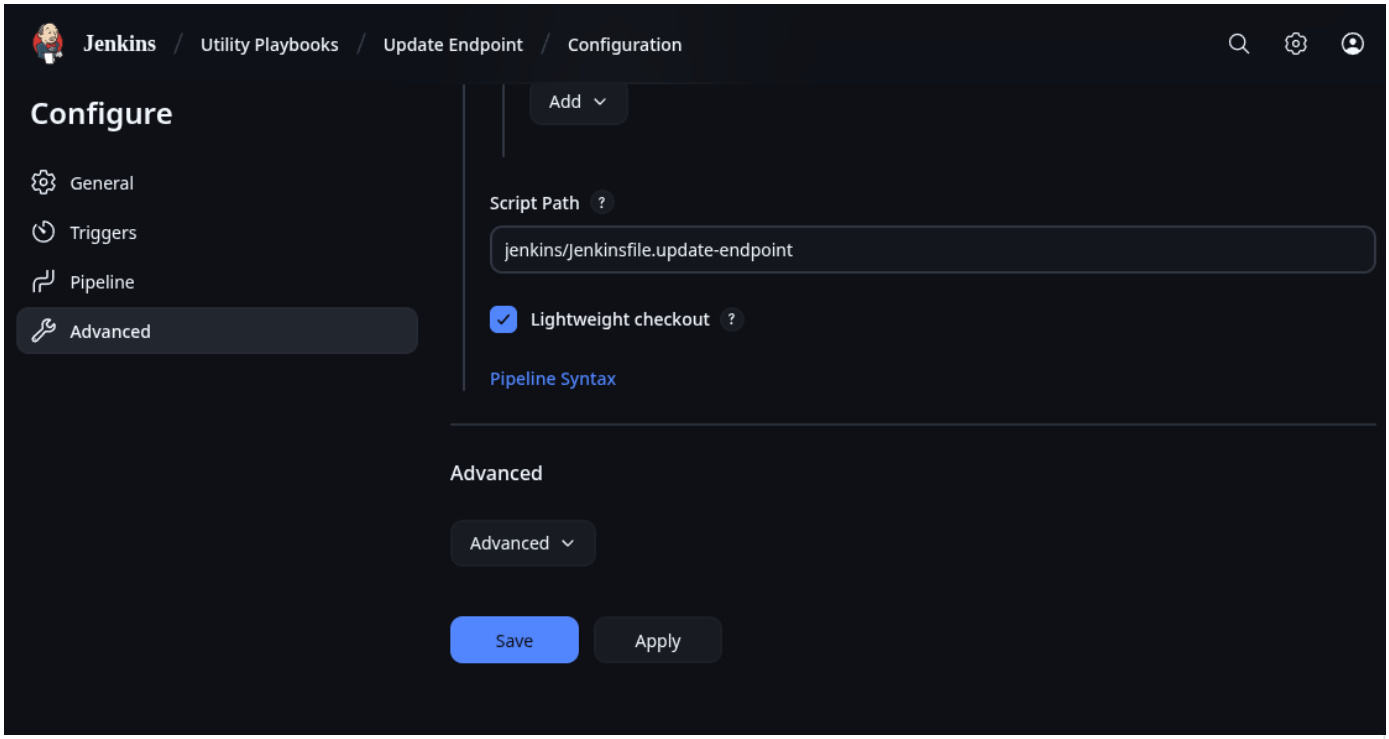
Ansible/Jenkins Code Server



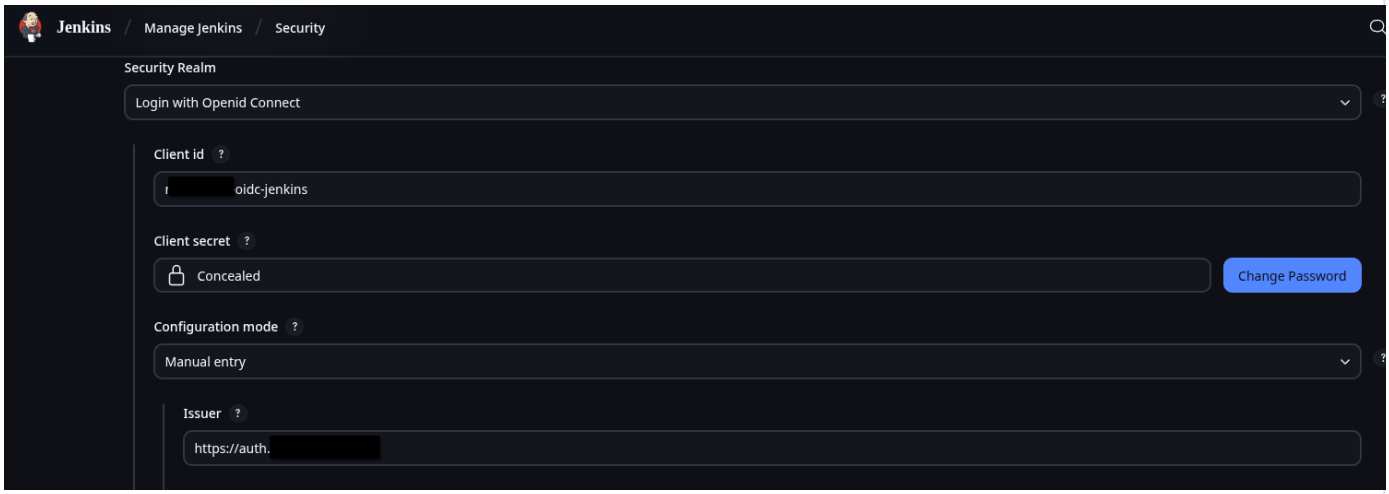
Jenkins Home Page

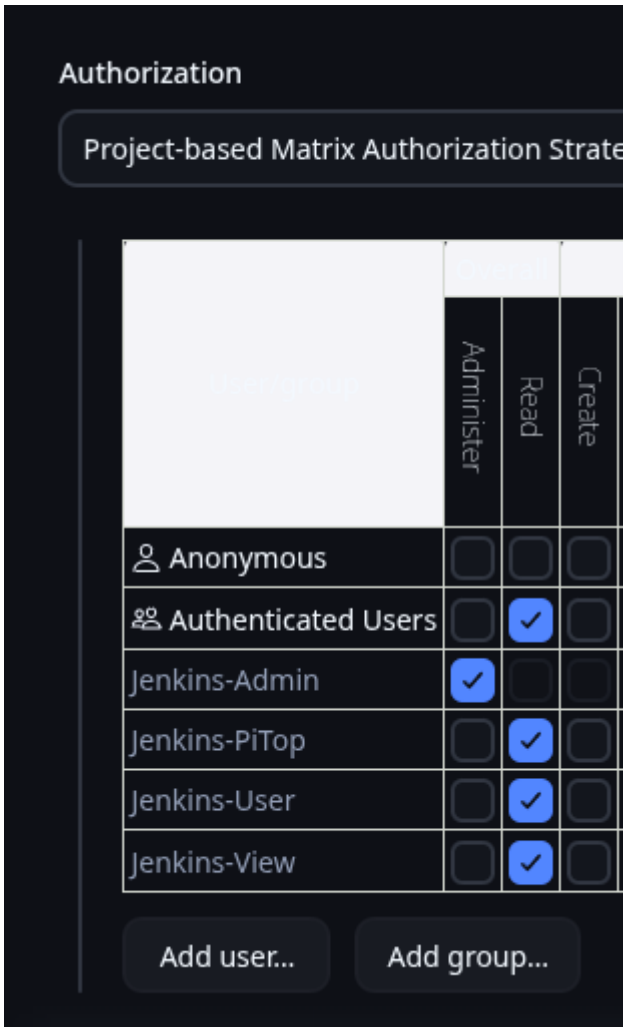


Sample Pipeline Config



Some OIDC Config and Security Settings





Let me start with my own Jenkins. I have recently shared my jenkinsfiles on my Gitea instance. To show how this works I will use the update-endpoint pipeline I have as an example. Most of my pipelines have the same stages of **Generate Inventory File**, **Ansible Playbook**, and **Remove Inventory File**. Once you have Jenkins up and running, and have stored your needed keys, you should be ready to run Ansible. The biggest thing is to make sure the SSH key is accessible. I will include the code for a dynamic inventory file generator here. This makes it obvious how Ansible authenticates. Since the inventory file is intended to be an **inventory**, it can be used to run the same playbook on multiple hosts.

inventory.sh

```
#!/bin/bash

# Dynamic inventory generation script ansible

# Function to display usage
usage() {
```

```
    echo "Usage: $0 -i IP_LIST -u JENKINS_USER -g JENKINS_GROUP [-s] [-v]"
    echo "Options:"
    echo "  -i IP_LIST       Comma-separated list of IPs"
    echo "  -u JENKINS_USER  Jenkins user for SSH access"
    echo "  -g JENKINS_GROUP Jenkins group for SSH access"
    echo "  -s               Set variable to true if more than one IP is passed"
    echo "  -v               Display Ansible Version"
    exit 1
}

# Initialize variables with default values
skip=false
more_than_one=false
display_version=false

# Parse command line options
while getopts ":i:u:g:sv" opt; do
  case ${opt} in
    i ) # process option i
        IP_LIST=$OPTARG
        ;;
    u ) # process option u
        JENKINS_USER=$OPTARG
        ;;
    g ) # process option g
        JENKINS_GROUP=$OPTARG
        ;;
    s ) # process option s
        skip=true
        ;;
    v ) # process option v
        display_version=true
        ;;
    \? ) usage
        ;;
  esac
done
shift $((OPTIND -1))
# Check if all required options are provided
```

```
if [ -z "$IP_LIST" ] || [ -z "$JENKINS_USER" ] || [ -z "$JENKINS_GROUP" ]; then
    usage
fi

if $display_version; then
    echo "Showing ansible version"
    ansible --version
fi

# Generate an 8-character hash from the IP list
hash=$(echo -n "$IP_LIST" | md5sum | cut -c 1-8)
echo "IP List:"
echo $IP_LIST
echo $hash

# Define the inventory file path with the hash
inventory_file="/var/jenkins_home/ansible/.inv/inventory-$hash.yml"

if $skip; then
    IFS=',' read -ra IPS <<< "$IP_LIST"
    if [ ${#IPS[@]} -gt 1 ]; then
        more_than_one=true
    fi
fi

if $skip; then
    echo "Single host option set"
    if $more_than_one; then
        echo "IP list provided, inventory will be emptied"
        IP_LIST=""
    fi
fi

# Initialize the YAML inventory content
inventory_content="---
all:
  hosts:
"
```

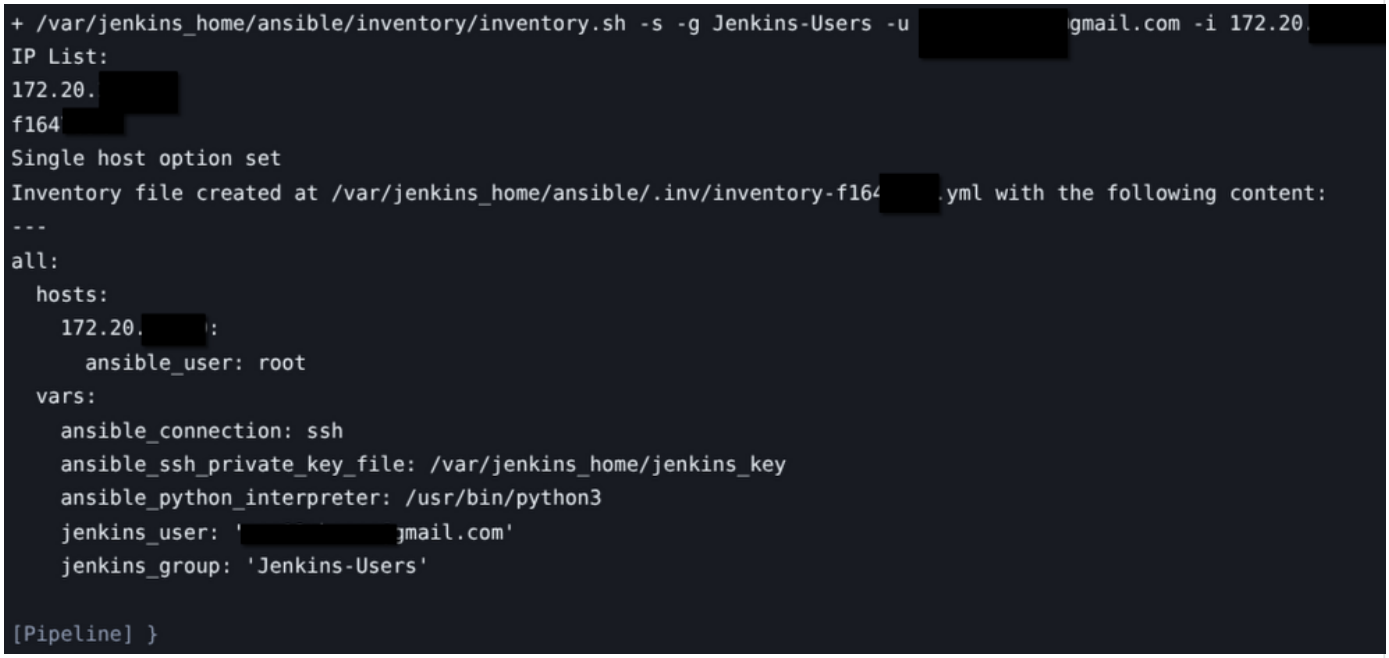
```
# Loop through each IP in the comma-separated list
IFS=',' read -ra IPS <<< "$IP_LIST"
for IP in "${IPS[@]}"; do
    inventory_content+="    ${IP}:
      ansible_user: root
"
done

inventory_content+="  vars:
    ansible_connection: ssh
    ansible_ssh_private_key_file: /var/jenkins_home/jenkins_key
    ansible_python_interpreter: /usr/bin/python3
    jenkins_user: '${JENKINS_USER}'
    jenkins_group: '${JENKINS_GROUP}'
"

# Write the inventory content to the file
echo "$inventory_content" > $inventory_file

echo "Inventory file created at $inventory_file with the following content:"
cat $inventory_file
```

This shows up in the pipeline run thusly:



After generating the inventory file, Ansible can just be ran by calling the playbook for the inventory. When you pick apart the Jenkinsfiles, the critical portion is this bit.

```
ansible-playbook -i \$inventory_file \
/var/jenkins_home/ansible/playbooks/update-endpoint.yaml --ssh-common-args='-o StrictHostKeyChecking=no'
```

You can see here how Ansible is called on the dynamically generated inventory on a particular playbook. Since this is the update playbook it is a simpler one, but it is still a playbook that runs a list of roles. This is a good transition to speaking about how Ansible talks about things. Ansible is the program, and the standard is that it runs Playbooks, which is a list of Roles, which themselves are lists of Tasks that run. There is a lot more technical stuff about Ansible including some magic word called **Idempotency**. The straight definition of Idempotency is the property of an operation that can be applied multiple times without changing the result beyond the initial execution. Effectively, this means that an Ansible playbook should have no effect when ran multiple times. Anyway, I just wanted to get that out of the way. I like to use Ansible for tinkering with Linux so that I don't have a result that I don't know how to reproduce. When I start my project in Ansible, I can keep track of every bit I need for it to work from start to finish, and if I screw up catastrophically I can just wipe it back to my base image.

Revision #9
Created 14 September 2025 20:14:32 by Matt Anderson
Updated 5 November 2025 17:01:56 by Matt Anderson