Ansible: Manifold Integration

David Harrigan / Product Updates Jul 9, 2019 Share

New in version 2.8, Ansible has added a built-in Manifold lookup plugin that allows you to get credentials from Manifold. This is really powerful if you're looking for a solution to securely manage your secrets and resources provisioned from Manifold in one place. In this post, I'll create a very simple Python application to show how easy it is to inject secrets from Manifold into your existing application via Ansible.

Embedded content: https://manifold.wistia.com/medias/xawimlt9d1

What is Ansible anyway?

Great question! Ansible is a server automation framework that allows you to provision an entire infrastructure, manage configuration, and deploy applications to your systems. It ain to be easy to use—the instructions (called playbooks) are written in YAML, and it relies on modules written in Python that implement the directives used in playbooks.

Before we start

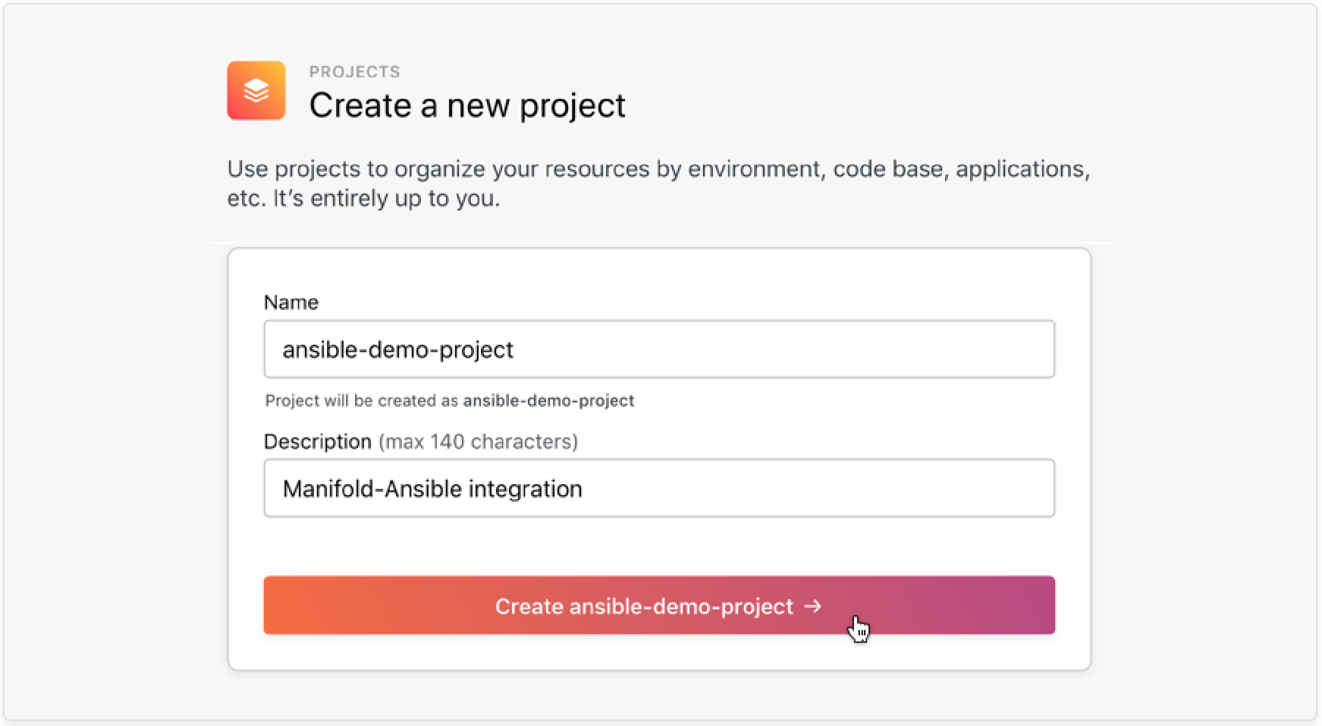
If you'd like to run through the example we show here, you'll need:

> 1. A Manifold account
> 2. Ansible 2.8 or higher
> 3. Vagrant
> 4. VirtualBox

Let's get Manifoldin'

Once you've logged in to Manifold.co, the first step is to create a new project and provision a new LogDNA resource. This will populate new credentials for accessing LogDNA under the selected project.

Here's what to do, step-by-step. First, create a new project.


PROJECTS

Create a new project

Use projects to organize your resources by environment, code base, applications, etc. It's entirely up to you.

Name

ansible-demo-project

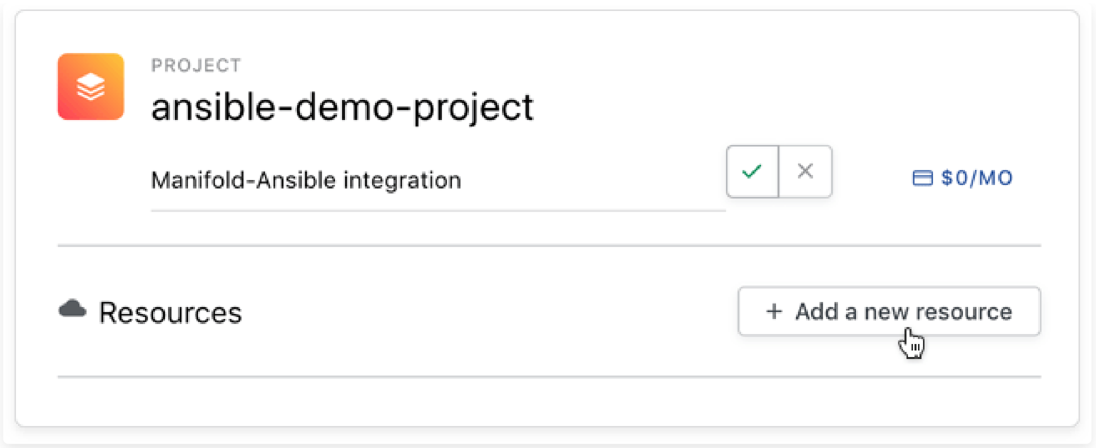
Project will be created as ansible-demo-project

Description (max 140 characters)

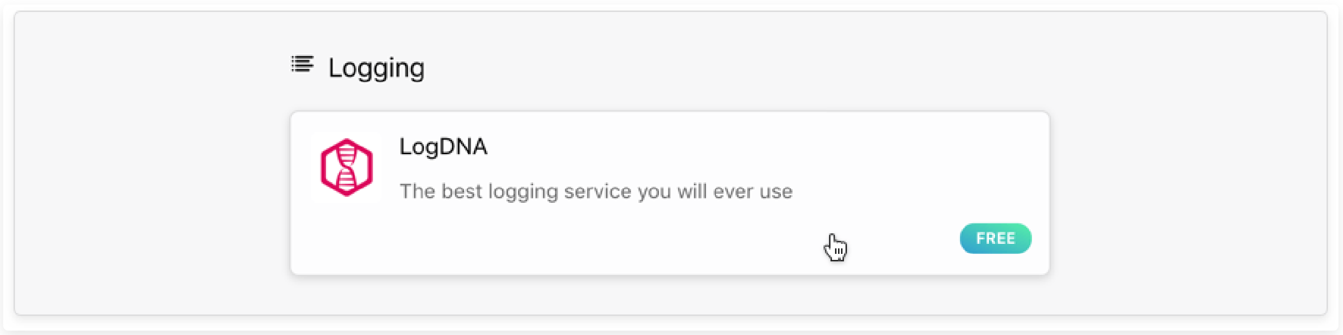
Manifold-Ansible integration

Create ansible-demo-project →

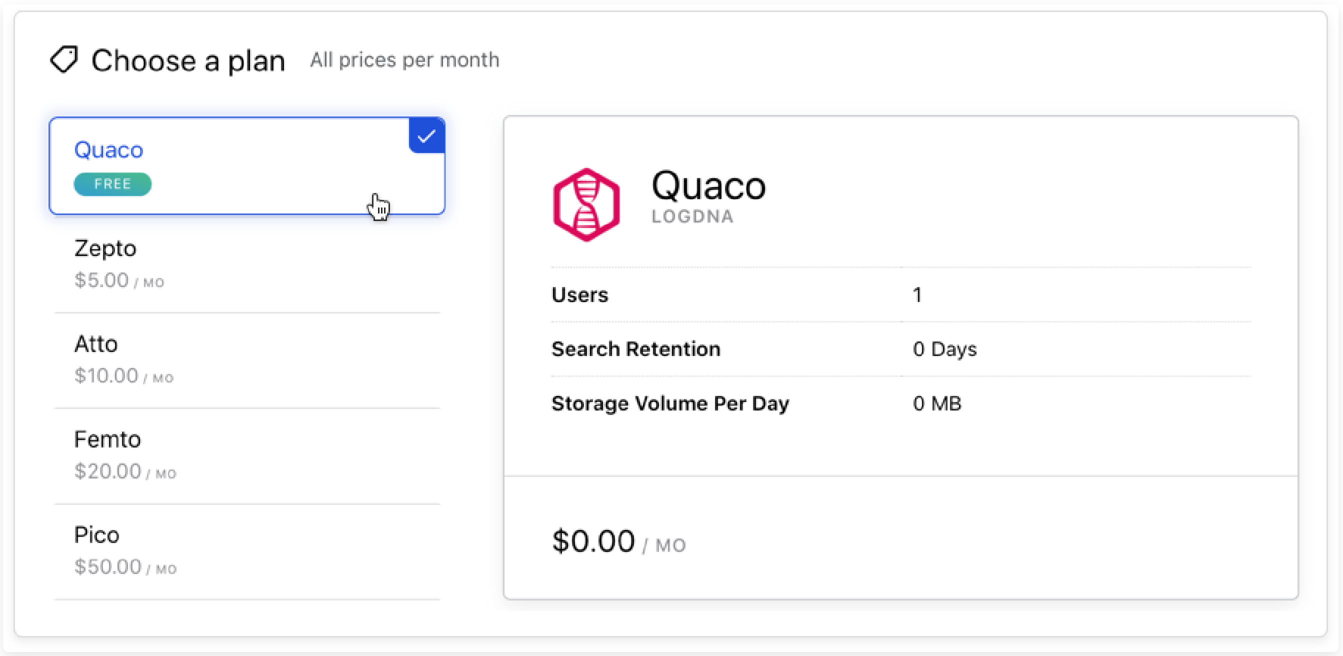
From your new project, click **Add a new resource**.



Scroll down and select **LogDNA**.



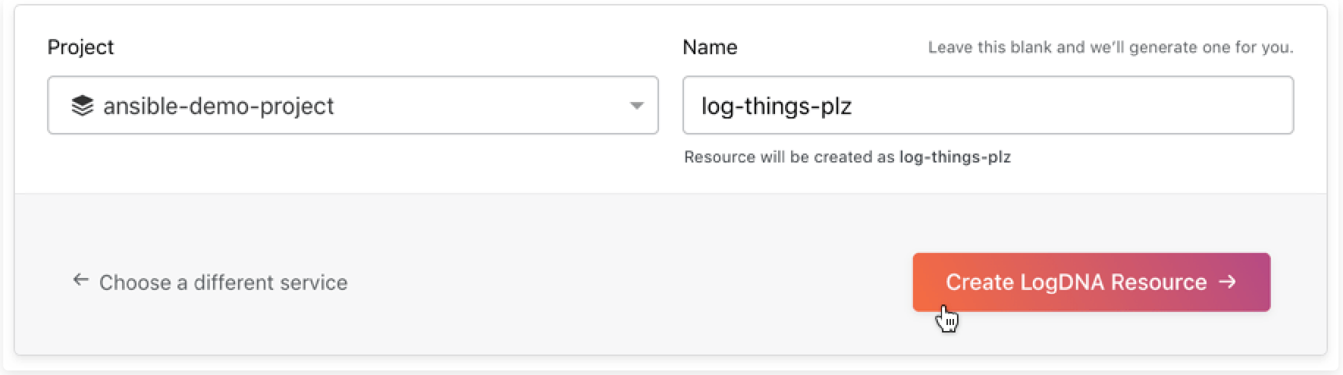
Keep the default free plan selection.



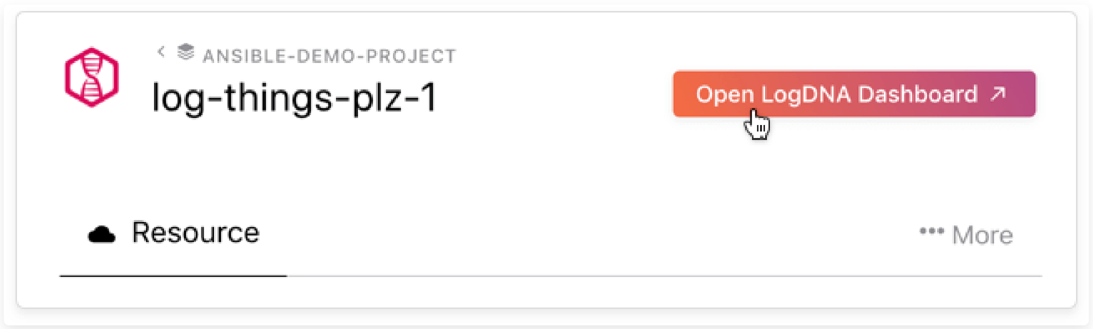
Users	1
Search Retention	0 Days
Storage Volume Per Day	0 MB

$0.00 / MO

Give your resource a unique name, and click **Create LogDNA Resource**.


Project
ansible-demo-project
Name
Leave this blank and we'll generate one for you.
log-things-plz
Resource will be created as log-things-plz
Choose a different service
Create LogDNA Resource

That's it! Now you have a cloud-based log aggregation and monitoring platform! If you're feeling spicy, click **Open LogDNA Dashboard**.



Be patient while your Manifold credentials are used for SSO access to LogDNA's dashboard. Spoiler: there won't be anything on the LogDNA dashboard for you to look at just yet, but we'll get to that soon.

A look at the sample files

Clone the sample ansible-demo repo. The repo contains a very simple Python app along with an Ansible playbook that provisions a VirtualBox VM to run the app. The app adds a LogDNA logging handler for our app to use, and the `health` endpoint simply logs a "health check" message every time you call `GET` on the `/health` endpoint.

The key required to push our logs to LogDNA will come in as an environment variable, and in our case, we'll let Ansible get this from Manifold, so that it's automatically injected during service start-up.

```
'logdna': {
  'level': 'DEBUG',
  'class': 'logging.handlers.LogDNAHandler',
  'key': os.environ.get('KEY'),
  'options': {
  'app': 'ansible-manifold-demo',
  },
},
```

In order to daemonize the very simple Python app, the repo sets up a systemd script to run the app as a systemd service. This means we need to inject our environment variable into the systemd service. We can use EnvironmentFile to let Ansible generate a new environment file for the systemd service to use.

```
- name: fetch credentials
  set_fact:
    manifold_secrets: "{{ lookup('manifold',
    'ansible-demo-logging', project='ansible-demo') }}"

- name: configure systemd env
  template:
    src: app.env.j2
    dest: /etc/systemd/system/app.env
  with_dict:
- "{{ manifold_secrets }}"
```

We use `set_fact` to save the result of `lookup` to the `manifold_secrets` variable. We then use it to create an `app.env` file using the `app.env.j2` template, which loops over the "manifold_secrets" dictionary.

```
{% for key, value in manifold_secrets.items() %}
{{ key }}={{ value }}
{% endfor %}
```

Depending on how your app is set up, it might make more sense to generate a Python file with the secrets embedded, directly expose the secrets as environment variables in your task, or store the secrets file in another directory with stricter permissions.

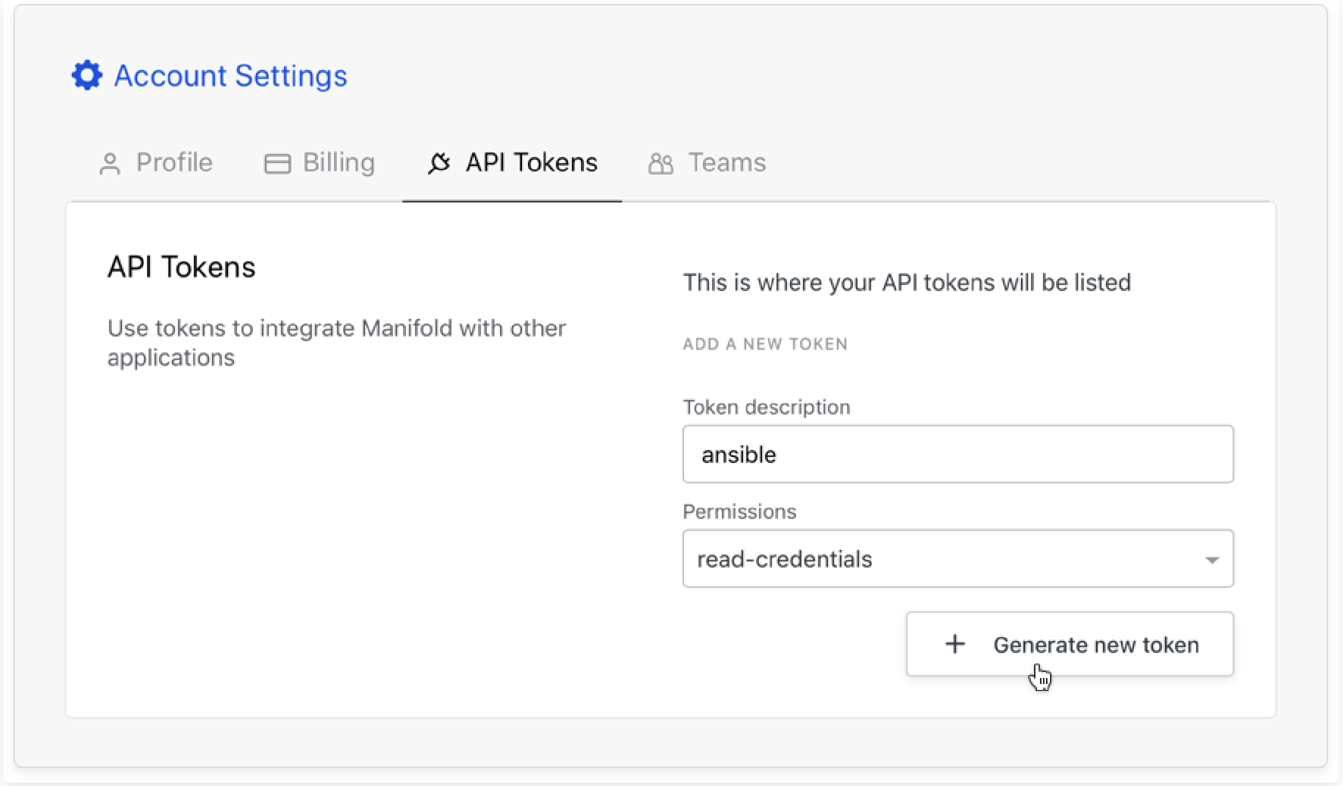
The rest of the playbook is straightforward — it installs some dependencies, copies the systemd config, and starts up the service. Here's what our entire playbook looks like:

```
- name: update apt cache if needed
  apt: update_cache=yes cache_valid_time=3600

- name: install pip
  apt:
    name: "python3-pip"
    state: latest
- name: install pip deps
  pip:
    requirements: /usr/src/app/requirements.txt

- name: install app systemd
  template:
    src: app.service.j2
    dest: /etc/systemd/system/app.service

- name: fetch credentials
  set_fact:
    manifold_secrets: "{{ lookup('manifold', 'ansible-demo-logging',
    project='ansible-demo') }}"

- name: configure systemd env
  template:
    src: app.env.j2
    dest: /etc/systemd/system/app.env
  with_dict:
    "{{ manifold_secrets }}"
```

Grab the keys to the kingdom

Before we can run the playbook and start provisioning, we need to obtain an API token from Manifold and expose it as an environment variable. We need this token to securely access credentials from Manifold.

Navigate to Manifold's API Tokens page and generate a token. Call it “ansible” and give it read-credentials permissions.



API Tokens

Use tokens to integrate Manifold with other applications

This is where your API tokens will be listed

ADD A NEW TOKEN

Token description: ansible

Permissions: read-credentials

Copy the generated token, and export it as `<strong>MANIFOLD_API_TOKEN` in your shell.

```
export MANIFOLD_API_TOKEN=<your API token>
```

Let’s run that playbook

Now we’re ready to run our playbook against a real machine! For simplicity, the repo includes a Vagrantfile which sets up an Ubuntu box using the VirtualBox provider and runs the playbook.

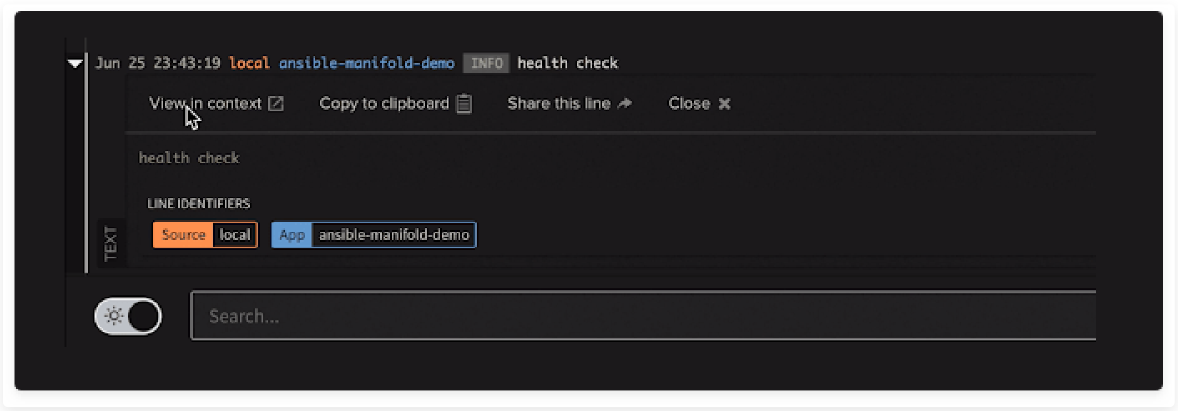
Run `vagrant up`. After a little bit, your VM should be running under the IP configured in the Vagrantfile.

Go ahead and curl the `health` endpoint from your host.

```
    curl -v 192.168.33.27:8000/health
    *   Trying 192.168.33.27...
    * TCP_NODELAY set
    * Connected to 192.168.33.27 (192.168.33.27) port 8000 (#0)
> GET /health HTTP/1.1
> Host: 192.168.33.27:8000
> User-Agent: curl/7.54.0
> Accept: */*
>
```

```
    * HTTP 1.0 assume close after body
< HTTP/1.0 204 No Content
< Date: Mon, 17 2019 21:24:51 GMT
< Server: WSGIServer/0.2 CPython/3.7.3
< Content-Length: 0
<
    * Closing connection 0
```

You should soon start seeing your logs in the LogDNA dashboard, which you can access from your LogDNA resource page in the Manifold dashboard.



```
Jun 25 23:43:19 local ansible-manifold-demo INFO health check
health check
```

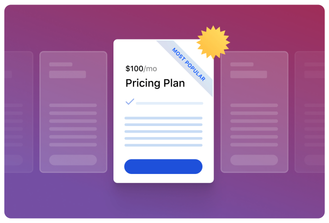
Wrapping it up

I hope you're as excited as we are about the new Ansible Manifold integration. You can provision managed cloud services as well as set up configuration and secrets independent of the tools you're using to manage your infrastructure — all you need is love and Manifold :D.

RECENT POSTS

Add Value to Your Service by Joining a Marketplace

Scott



Comparing the Top 5 Pricing Models for Developer Apps by Fitzpatrick

by Chris Tozzi