

Ampere Computing

# Getting Cloud-Native with FreeBSD in OCI using Ampere A1 and Terraform

## Table of Contents

* [Introduction](#introduction)
* [Requirements](#requirements)
* [Using the oci-ampere-a1 terraform module](#using-the-oci-ampere-a1-terraform-module)
  + [Configuration with terraform.tfvars](#configuration-with-terraformtfvars)
  + [Creating the main.tf](#creating-the-maintf)
  + [Creating a cloud-init template](#creating-a-cloud-init-template)
  + [Running Terraform](#running-terraform)
  + [Logging in](#logging-in)
  + [Destroying when done](#destroying-done)

## Introduction

Here at [Ampere Computing](https://amperecomputing.com) we are always interested in diverse workloads for our cloud-native Ampere(R) Altra(TM) Aarch64 processors, and that includes down to the the choose of your Operating System. [FreeBSD](https://freebsd.org) is an active open source software project originally developed over 30 years ago focusing on features, speed, and stability. It is derived from BSD, the version of UNIX® developed at the University of California, Berkeley for servers, desktops and embedded systems. It has been a staple in datacenters for years due to it’s advanced networking, security and storage features, which have also made it a choice for powering diverse platforms amoungst some of the largest web and internet service providers.

For those unfamiliar with [FreeBSD](https://freebsd.org), it has similarities with Linux, with similar package management tooling and methods, packages, and open source software stacks available for installation easily.

| — | FreeBSD | Linux |
| --- | --- | --- |
| User Experience | package management, shells, remote management, compilers and development platforms | package management, shells, remote management, compilers and development platforms |
| Project Scope | The [FreeBSD](https://freebsd.org) Project maintains a complete system. The project community delivers a kernel, device drivers, userland utilities, and documentation. | Linux community on the other hand focuses on only delivering a kernel and drivers. The Linux Community relies on third-parties like Enterprise Linux vendors, and other Open Source Linux OS projects for system software to be curated. |
| Licensing | FreeBSD source code is generally released under a permissive BSD license. The kernel code and most newly created code are released under the two-clause BSD license which allows everyone to use and redistribute FreeBSD as they wish. In general a lax, permissive non-copyleft free software license, compatible with the GNU GPL. The BSD license is intended to encourage product commercialization. BSD-licensed code can be sold or included in proprietary products without restriction on future behavior. | Linux is licensed with the GPL. The GPL, while designed to prevent the proprietary commercialization of open source code, is considered a copyleft license and can still provide strategic advantage to a company deciding to build solutions using Linux. |

When choosing a cloud-native OS you might not think of [FreeBSD](https://freebsd.org), however it supports the same industry standard metadata interfaces for instance configurations as Linux, [Cloud-Init](https://cloud-init.io). This allows you to automate your [FreeBSD](https://freebsd.org) workloads, in a simlar fashion to other operating system options. This meams [FreeBSD](https://freebsd.org) is perfectly suitable when using on a cloud platform.

Now personally speaking I have been working with the great team at the [FreeBSD](https://freebsd.org) project for some time watching thier craftmanship, curating, iterating, and helping achive the “it just works” experience for Aarch64 and Ampere platforms and customers who choose to build and run solutions on [FreeBSD](https://freebsd.org). Recently [FreeBSD](https://freebsd.org) became available for use on Ampere A1 shapes within the [Oracle OCI](https://www.oracle.com/cloud/free/#always-free) marketplace.

In this post, we will build upon prevous work to quickly automate using [FreeBSD](https://freebsd.org) on Ampere(R) Altra(TM) Arm64 processors within Oracle Cloud Infrastructure using Ampere A1 shapes.

## Requirements

Obviously to begin you will need a couple things. Personally I’m a big fan of the the DevOPs tools that support lots of api, and different use cases. [Terraform](https://www.terraform.io/downloads.html) is one of those types of tools. If you have seen my [prevous session with some members of the Oracle Cloud Infrastracture team](https://youtu.be/3F5EnHRPCI4), I build a terraform module to quickly get you started using Ampere plaforms on OCI. Today we are going to use that module to launch a [FreeBSD](Instance) virtual machine while passing in some metadata to configure it.

* [Terraform](https://www.terraform.io/downloads.html) will need be installed on your system.
* [Oracle OCI “Always Free” Account](https://www.oracle.com/cloud/free/#always-free) and credentials for API use

## Using the oci-ampere-a1 terraform module

The [oci-ampere-a1](https://github.com/amperecomputing/terraform-oci-ampere-a1) terraform module code supplies the minimal ammount of information to quickly have working Ampere A1 instances on OCI [“Always Free”](https://www.oracle.com/cloud/free/#always-free). It has been updated to include the ability to easily select [FreeBSD](https://freebsd.org) as an option. To keep things simple from an OCI perspective, the root compartment will be used (compartment id and tenancy id are the same) when launching any instances. Addtional tasks performed by the [oci-ampere-a1](https://github.com/amperecomputing/terraform-oci-ampere-a1) terraform module.

* Operating system image id discovery in the user region.
* Dynamically creating sshkeys to use when logging into the instance.
* Dynamically getting region, availability zone and image id.
* Creating necessary core networking configurations for the tenancy
* Rendering metadata to pass into the Ampere A1 instance.
* Launch 1 to 4 Ampere A1 instances with metadata and ssh keys.
* Output IP information to connect to the instance.

### Configuration with terraform.tfvars

For the purpose of this we will quickly configure Terraform using a terraform.tfvars in the project directory.  
Please note that Compartment OCID are the same as Tenancy OCID for Root Compartment. The following is an example of what terraform.tfvars should look like:

tenancy\_ocid = "ocid1.tenancy.oc1..aaaaaaaabcdefghijklmnopqrstuvwxyz1234567890abcdefghijklmnopq"  
user\_ocid = "ocid1.user.oc1..aaaaaaaabcdefghijklmnopqrstuvwxyz0987654321zyxwvustqrponmlkj"  
fingerprint = "a1:01:b2:02:c3:03:e4:04:10:11:12:13:14:15:16:17"  
private\_key\_path = "/home/bwayne/.oci/oracleidentitycloudservice\_bwayne-08-09-14-59.pem"

For more information regarding how to get your OCI credentials please refer to the following reading material:

* <https://docs.oracle.com/en-us/iaas/Content/API/SDKDocs/terraformproviderconfiguration.htm>
* [Where to Get the Tenancy’s OCID and User’s OCID](https://docs.oracle.com/en-us/iaas/Content/API/Concepts/apisigningkey.htm#five)
* [API Key Authentication](https://docs.oracle.com/en-us/iaas/Content/API/SDKDocs/terraformproviderconfiguration.htm#APIKeyAuth)
* [Instance Principal Authorization](https://docs.oracle.com/en-us/iaas/Content/API/SDKDocs/terraformproviderconfiguration.htm#instancePrincipalAuth)
* [Security Token Authentication](https://docs.oracle.com/en-us/iaas/Content/API/SDKDocs/terraformproviderconfiguration.htm#securityTokenAuth)
* [How to Generate an API Signing Key](https://docs.oracle.com/en-us/iaas/Content/API/Concepts/apisigningkey.htm#two)

### Creating the main.tf

To use the terraform module you must open your favorite text editor and create a file called main.tf. Copy the following is code to supply a custom cloud-init template:

variable "tenancy\_ocid" {}  
variable "user\_ocid" {}  
variable "fingerprint" {}  
variable "private\_key\_path" {}  
  
locals {  
 cloud\_init\_template\_path = "${path.cwd}/cloud-init.yaml.tpl"  
}  
  
module "oci-ampere-a1" {  
 source = "github.com/amperecomputing/terraform-oci-ampere-a1"  
 tenancy\_ocid = var.tenancy\_ocid  
 user\_ocid = var.user\_ocid  
 fingerprint = var.fingerprint  
 private\_key\_path = var.private\_key\_path  
# Optional  
# oci\_vcn\_cidr\_block = "10.2.0.0/16"  
# oci\_vcn\_cidr\_subnet = "10.2.1.0/24"  
 oci\_os\_image = "freebsd"  
 instance\_prefix = "ampere-a1-freebsd"  
 oci\_vm\_count = "1"  
 ampere\_a1\_vm\_memory = "24"  
 ampere\_a1\_cpu\_core\_count = "4"  
 cloud\_init\_template\_file = local.cloud\_init\_template\_path  
}  
  
output "oci\_ampere\_a1\_private\_ips" {  
 value = module.oci-ampere-a1.ampere\_a1\_private\_ips  
}  
output "oci\_ampere\_a1\_public\_ips" {  
 value = module.oci-ampere-a1.ampere\_a1\_public\_ips  
}

### Creating a cloud init template.

Using your favorite text editor create a file named cloud-init.yaml.tpl in the same directory as the main.tf you previously created. Copy the following content into the text file and save it.

#cloud-config  
package\_update: true  
package\_upgrade: true  
packages:  
 - tmux  
 - rsync  
 - git  
 - curl  
 - bzip2  
 - python3  
 - python3-devel  
 - python3-pip-wheel  
 - gcc  
 - gcc-c++  
 - bzip2  
 - screen  
groups:  
 - ampere  
system\_info:  
 default\_user:  
 groups: [ampere]  
runcmd:  
 - echo 'OCI Ampere FreeBSD Example' >> /etc/motd

### Running Terraform

Executing terraform is broken into three commands. The first you must initialize the terraform project with the modules and necessary plugins to support proper execution. The following command will do that:

terraform init

Below is output from a ‘terraform init’ execution within the project directory.

After ‘terraform init’ is executed it is necessary to run ‘plan’ to see the tasks, steps and objects. that will be created by interacting with the cloud APIs. Executing the following from a command line will do so:

terraform plan

The ouput from a ‘terraform plan’ execution in the project directy will look similar to the following:

Finally you will execute the ‘apply’ phase of the terraform exuction sequence. This will create all the objects, execute all the tasks and display any output that is defined. Executing the following command from the project directory will automatically execute without requiring any additional interaction:

terraform apply -auto-approve

The following is an example of output from a ‘apply’ run of terraform from within the project directory:

### Logging in

Next you’ll need to login with the dynamically generated sshkey that will be sitting in your project directory. To log in take the ip address from the output above and run the following ssh command:

ssh -i ./oci-is\_rsa freebsd@155.248.228.151

You should be automatically logged in after running the the command. The following is output from sshing into an instance and then running ‘sudo cat /var/log/messages’ to verify cloud-init execution and package installation:

### Destroying when done

You now should have a fully running and configured FreeBSD instance. When finished you will need to execute the ‘destroy’ command to remove all created objects in a ‘leave no trace’ manner. Execute the following from a command to remove all created objects when finished:

terraform destroy -auto-approve

The following is example output of the ‘terraform destroy’ when used on this project.

Modifing the cloud-init file and then performing the same workflow will allow you to get interating quickly. At this point you should definately know how to quickly get automating using FreeBSD with Ampere on the Cloud!