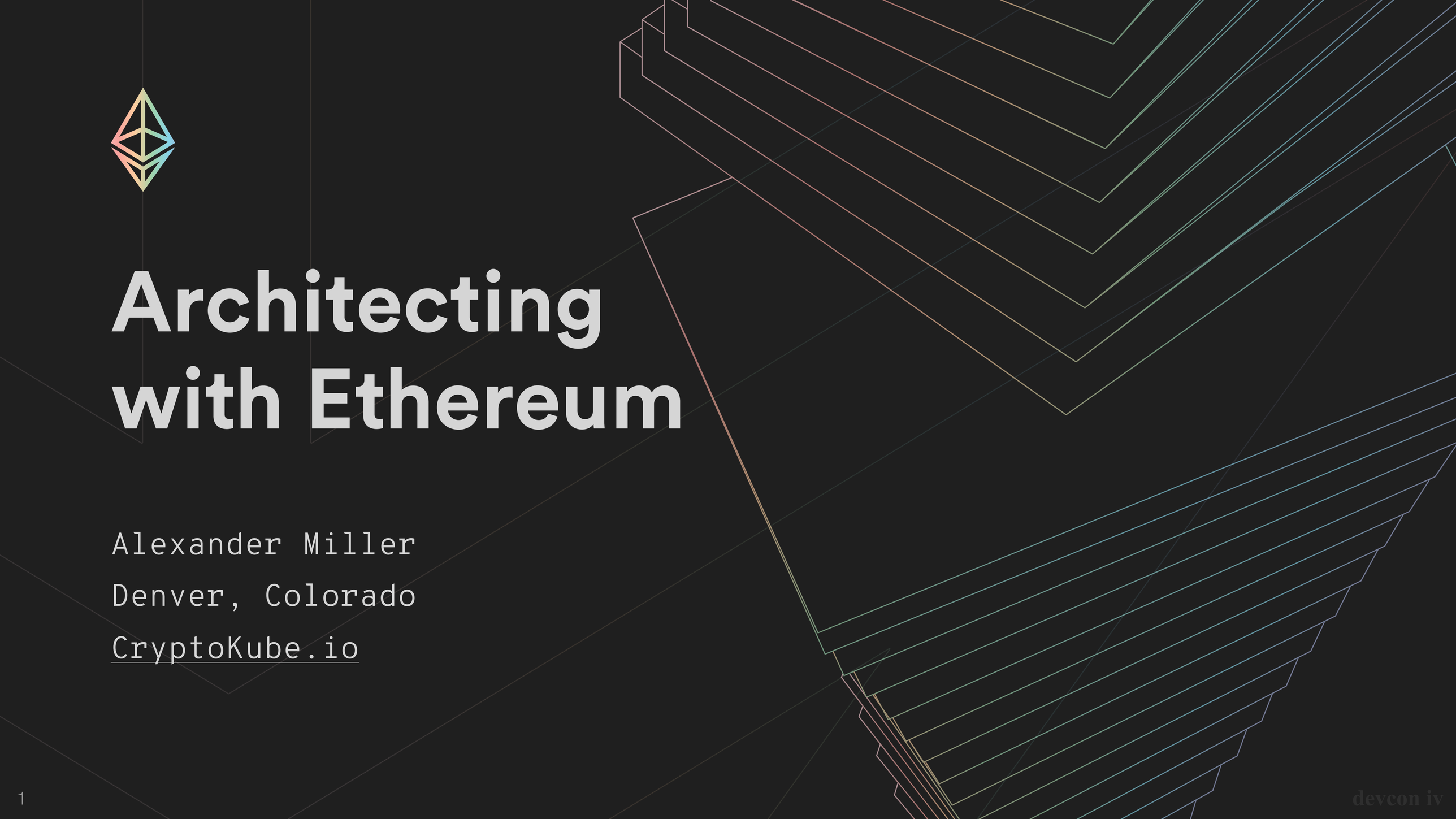


Architecting with Ethereum

Alexander Miller
Denver, Colorado
CryptoKube.io

Introduction

Intro: Who am I?

- Based out of Denver, Colorado
- 2012: discovered Bitcoin, started mining w/GPU -> BFL ASIC
- 2014-2016: platform support engineer @ Hortonworks (Hadoop)
- 2016-2017: tooling engineer @ Endless OS (Linux distro)
- 2017: discovered Ethereum, devops @ Shapeshift
- 2018: open source sabbatical, distributed systems consulting
- Passions: open source, P2P crypto, radical self-reliance

Intro: Workshop

GOAL: build an Ethereum computing stack from re-usable modules

AGENDA:

- Introduce admin tooling & major stack components
- Conduct a series of exercises to demonstrate the concepts in realistic use cases. Each exercise:
  - builds on previous exercises
  - introduces one admin concept & one Ethereum node concept

Intro: CryptoKube

- FOSS platform for hosting P2P crypto applications
- Easily provision & deploy components to multiple environments
- Currently a collection of Ansible roles & Terraform modules
- Integrated stack is in development
- Latest details at https://www.cryptokube.io

Intro: Prerequisites

- **Experience**: remote Linux administration, and high-level understanding of Ethereum and cloud computing concepts
- **Software**: SSH client & web browser
- **DigitalOcean**:
  - Create account (use link for $100/60-day trial)
  - Add SSH public key for authentication on new droplets
  - Create a personal access token for API access

Intro: DigitalOcean setup

Setup guide:

https://www.cryptokube.io

Administrative Tooling

Administrative Tooling

- Terraform
- Ansible
- Git
- Docker
- DigitalOcean

Administrative Tooling
Terraform

Administrative Tooling: Terraform

- Codifies APIs into declarative configuration files that can be shared, edited, reviewed, and versioned (just like code)
- Safely and predictably create, change, and improve infrastructure
- Version control your resources, allowing rollback to previous state in case of error
- Uses declarative syntax (HCL), fully JSON compatible but extended for easier human consumption
- Maintained by Hashicorp. https://www.terraform.io

Administrative Tooling: Terraform concepts

Configuration

Describes infrastructure and sets variables. Stored as text files with **.tf** extension.

State

Maps real world resources to your configuration, and keeps track of metadata. Stored locally as **terraform.tfstate** (or remotely)

Providers

Responsible for understanding API interactions and exposing resources. Generally an IaaS (AWS, DigitalOcean), PaaS (Heroku), or SaaS (CloudFlare).

Modules

Self-contained packages of Terraform configs that are managed as a group. Used to create reusable components and for basic code organization.

Administrative Tooling: Terraform commands

- terraform **init** - Initialize a new or existing Terraform configuration (install plugins, perform minimal validation)
- terraform **get** - Import modules
- terraform **plan** - Generate and show an execution plan
- terraform **show** - Inspect Terraform state or plan
- terraform **apply** - Builds or changes infrastructure
- terraform **destroy** - Destroy Terraform-managed infrastructure

Administrative Tooling
Ansible

Administrative Tooling: Ansible

- Simple IT automation engine that automates configuration management, application deployment, and more
- Native OpenSSH remote communication when possible (no agents)
- Executes ad-hoc commands or scripted “playbooks”
- Extensible via modules, plugins, and Python APIs
- Maintained by Red Hat. https://www.ansible.com

Administrative Tooling: Ansible concepts

Inventory

Consists of hosts and groups. May also define variables. We will generate ours from Terraform state using **terraform-inventory**

Playbooks

Consists of plays, which map tasks and roles to a group of hosts from the inventory. YAML format.

Roles

Collections of related variables, tasks, templates, and files. Primary mechanism for breaking playbooks into reusable components.

Modules

Reusable, standalone scripts that can be used by Ansible. Return information as JSON to stdout. Take arguments in several ways. See ‘Module Index’

Administrative Tooling: Ansible Inventory

```
[ethereum_nodes]
geth_nodes
parity_nodes

[geth_nodes]
geth-light.example.com geth_sync_mode=light
geth-full[0:2].example.com

[parity_nodes]
parity-full.example.com
```

Administrative Tooling: Ansible Playbooks

```
---
- hosts: ethereum_nodes
  vars:
    data_dir: ‘/opt/.ethereum’
    rpc_port: 8545
  tasks:
  - include_role:
      name: rhel_common
    when: "ansible_facts['os_family'] == 'RedHat'"
  - include_role:
      name: debian_common
    when: "ansible_facts['os_family'] == 'Debian'"
```

Administrative Tooling: Ansible Modules

A few quick examples:

- **- lineinfile:** path=/etc/selinux/config regexp=‘^SELINUX='
  line=‘SELINUX=enforcing'

- **- template:**
    src: etc/ssh/sshd_config.j2
    dest: /etc/ssh/sshd_config
    owner: root
    group: “{{sshd_group}}”
    mode: ‘0600'

Administrative Tooling: Ansible Roles

Roles contain some (or all) of these subdirectories:

- **tasks** – list of tasks to be executed
- **handlers** – special tasks that can be triggered to run at end
- **defaults** – default variables for the role
- **vars** – other variables for the role
- **files** – static files that can be copied
- **templates** – dynamic files that can be altered by variables
- **meta** – role dependencies, author info, etc

Administrative Tooling: Ansible commands

- Ad-hoc command to display Linux distribution info:
  `ansible all -a 'cat /etc/lsb_release'`
- Run playbook 'site.yml' against Terraform hosts:
  `ansible-playbook -i terraform-inventory site.yml`
- Install Galaxy-style roles specified in requirements.yml:
  `ansible-galaxy install -r requirements.yml -f`
- Generate scaffolding for new Galaxy-style role:
  `ansible-galaxy init <role-name>`

Administrative Tooling DigitalOcean

Administrative Tooling: DigitalOcean

- Infrastructure as a Service (IaaS) platform
- Powerful & simple API, friendly web UI
- Extensive set of documentation & tutorials
- Cloud resources include:
  - Servers, Block storage, Object storage, Machine images
  - Firewalls, Load balancers, Floating IPs, DNS

Administrative Tooling: additional resources

- Git – distributed version-control system
- Docker – popularized containerization, isolate components
- Packer – build images for cloud, VMs, containers
- Consul – service discovery & KV store

Infrastructure Components

Infrastructure Components

- Ethereum Node
- Application Server
- Reverse Proxy

Infrastructure Components: Ethereum Node

Ethereum Node

- Services & ports
- Networks & chains
- Node types
  - Light client
  - Full node
- Implementations
  - Geth
  - Parity

Ethereum Node: services & ports

Service	Protocol	Port	Interface
JSON-RPC	TCP	8545	private
Websockets	TCP	8546	private
Peer-to-peer	TCP	30303	public
Node discovery	UDP	30301	public

Ethereum Node: networks & chains

Network ID	Chain ID	Description	Consensus	Clients
1	1	Ethereum mainnet	PoW	All
3	1	Ropsten testnet	PoW	All
4	1	Rinkeby testnet	PoA	Geth
42	1	Kovan testnet	PoA	Parity
1	61	Ethereum Classic mainnet	PoW	All
2	1	Ethereum Classic testnet	PoW	All

Ethereum Node: synchronization

- **Chain data**: list of blocks containing transactions
- **State data**: results of each transaction's state transition
  - Account balances, nonces, smart contract code and data
  - Importing state entries now takes much longer than downloading the blocks
- A **full node** stores all blocks and a pruned account state
- A **light client** only stores the headers of all blocks, and downloads state as needed

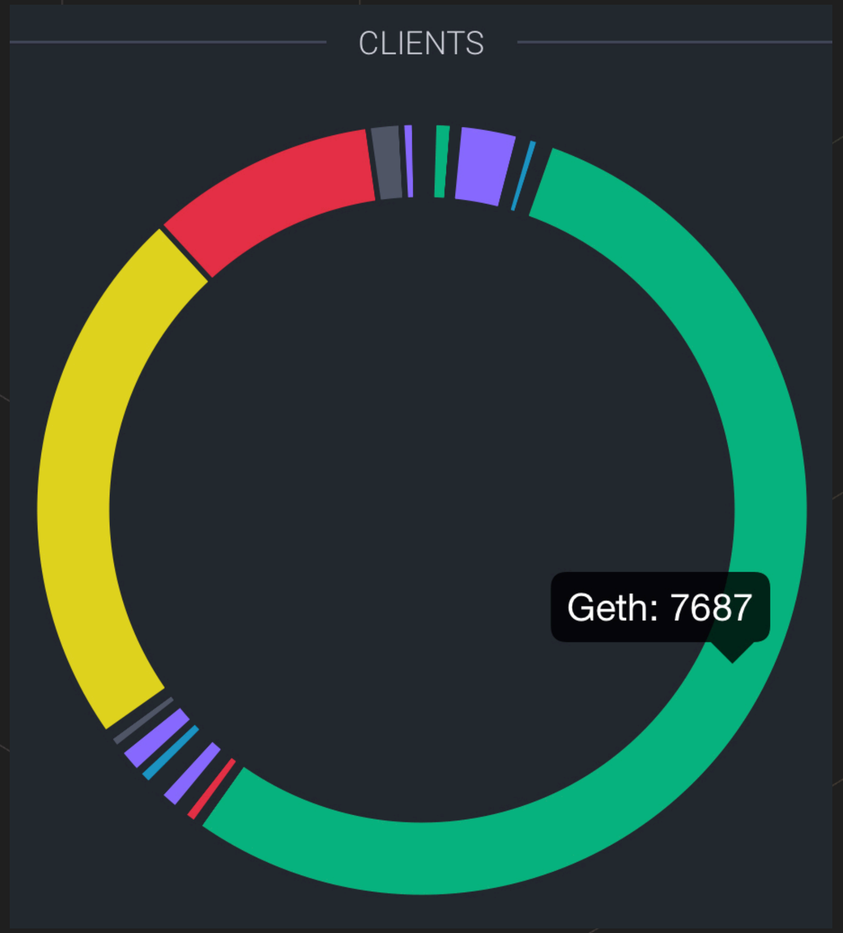
Ethereum Node: node types

all "full clients" except for archive nodes (intended to be run by businesses, block explorers, etc) will eventually be set up as "partially light clients" with respect to all history older than a few thousand blocks

- https://github.com/ethereum/wiki/wiki/Light-client-protocol

- **Full node** – download all blocks, keep pruned state
- **Light client** – download block headers, state on-demand
- **Archive node** – download all blocks, keep unpruned state
- **Bootstrap node** – no blocks, provides directory of other nodes

Ethereum Node: implementations



EtherNodes.org

An Ethereum client implements the protocol specified in the yellow paper.

Many implementations are available, but the vast majority of mainnet is running Geth or Parity.

Ethereum Node: Geth (go-ethereum)

- Written in Go
- Developed by Ethereum Foundation
- Licensed under GPLv3
- Separate implementation for each chain (ETH, ETC, Ubiq, etc)
- Best configured using only command-line arguments

Ethereum Node: Parity Ethereum

- Written in Rust
- Developed by Parity Tech (UK)
- Licensed under GPLv3
- Single implementation supports many networks & chains
- Configurable by file and/or command-line arguments

Ethereum Node: parity (additional projects)

- **Feather** - wallet for ETH & ERC-20 that runs on top of Parity Ethereum light client
- **Signer** - mobile app to use smartphone as air-gapped wallet for cold storage
- **Parity UI** (deprecated) - previously accessible as a web app from the Ethereum client
- **Substrate** - framework for building new blockchains

Infrastructure Components: Application Server

Infrastructure Components: Application Server

- Act as an Ethereum remote client, which relies on a node to provide access to the blockchain
- May interact with a light client running locally on the host
- May interact with other P2P/crypto applications on the host

Infrastructure Components: Reverse Proxy

Infrastructure Components: Reverse Proxy

- **Load balancing** improves the performance and reliability of a service by distributing the workload across multiple computing resources, such as hosts or network links.
- **High availability** assures a high level of operational performance for a given period of time by eliminating single points of failure
- **TLS Termination** provides a secure connection to the outside
- Implemented using HAProxy, NGINX, or a cloud mechanism

Infrastructure Components: HAProxy

- Dedicated reverse proxy component
- Layer 3 (TCP) and Layer 7 (HTTP) load balancing
- Health checking
- Rate limiting
- SSL/TLS termination
- Multiple instances can use a heartbeat for high availability
- Very fast and efficient

Infrastructure Components: NGINX

- Primarily a web server, can serve static and dynamic content
- Easily configured as a reverse proxy
- Also able to perform caching duties
- Status page only provides 7 metrics (versus 61 for HAProxy)

Workshop Setup

Workshop Setup: DigitalOcean account

- Create account (use link for $100/60-day trial)
  https://do.co/devcon4-eth
- Add SSH public key for authentication on new droplets
- Create a personal access token for API access
- **QuickStart Guide: https://www.cryptokube.io**

Workshop Setup: Management Host

Create a droplet for the **Management Host** using these options:

- **Image:** Ubuntu 18.04 x64
- **Size:** 2GB/2vCPU standard droplet
- **Datacenter region:** your choice
- **Additional options:** private networking, user data, monitoring
  - **User data**: see contents on next slide (**docs**)
- **SSH keys:** select yours
- click **Create**

Workshop Setup: User-Data

```
#cloud-config
# from: https://cryptokube.io/devcon4/cloud-config.yaml

package_upgrade: true

packages:
  - python
  - python-pip
  - git
  - zip
  - jq

runcmd:
  - [curl, -o, /tmp/terraform.zip, “https://releases.hashicorp.com/terraform/0.11.10/
terraform_0.11.10_linux_amd64.zip"]
  - [unzip, -d, /usr/local/bin/, /tmp/terraform.zip]
  - [curl, -L, -o, /tmp/terraform-inventory.zip, "https://github.com/adammck/
terraform-inventory/releases/download/v0.7-pre/terraform-inventory_v0.7-
pre_linux_amd64.zip"]
  - [unzip, -d, /usr/local/bin/, /tmp/terraform-inventory.zip]
  - [pip, install, -U, pip, ansible]
  - [git, clone, “https://github.com/cryptokube-io/devcon4-workshop.git”]
```

Workshop Setup: completion

- **SSH into the management host**

  ssh root@<ip>

- **Monitor cloud-init progress**

  tail -f /var/log/cloud-init-output.log

- **Enter workshop directory**

  cd devcon4-workshop

- **Run initialization script**

  bin/init_config

Workshop Exercises

Workshop Exercises

- 01 Light client
- 02 Full node
- 03 Reverse proxy
- 04 Ethereum Application

Exercise 01
Light Client

Exercise 01: Light Client

- **Introduces**: digitalocean, parity, light client
- **Goal**: deploy a light client on a DO droplet using Terraform
- **Terraform config**: provisions a single DO droplet
- **Ansible playbook**: installs & configures Parity Ethereum
- This is mainly to verify that the cloud admin tools are working. It is more important to observe the commands and output than it is to get the node synced.

Exercise 01: Terraform config

Image: Ubuntu 18.04 x64

Name: devcon4-parity_light

Ports (proto int:ext ip - name)

- TCP 8545:8545 127.0.0.1 - HTTP RPC
- TCP 8546:8546 127.0.0.1 - HTTP WS
- TCP 30303:30303 0.0.0.0 - P2P
- UDP 30301:30301 0.0.0.0 - node discovery

Exercise 01: [1/3] build the infrastructure

- **Enter the exercise directory**

  ```
  cd 01_light_client
  ```

- **Initialize the Terraform configuration**

  ```
  terraform init
  ```

- **Get Terraform modules**

  ```
  terraform get
  ```

- **View the Terraform execution plan**

  ```
  terraform plan
  ```

- **Apply the Terraform config to build the infrastructure**

  ```
  terraform apply
  ```

Exercise 01: [2/3] run Ansible playbook

- **Run Ansible playbook**

  ```
  ansible-playbook -i terraform-inventory site.yml
  ```

Exercise 01: [3/4] query the node

Query the node using the JSON-RPC API

```
ip=$(terraform-inventory -list | jq -r .parity_node[0])
header="Content-Type: application/json”
host="http://ip:8545"
queries=“web3_clientVersion net_version net_peerCount”

for q in queries; do
  data="{'jsonrpc':'2.0','method':'$q','params':[],'id':67}"
  curl -H "$header" -X POST -data “$data”
done
```

Exercise 01: [3/3] clean up

- **Clean up the infrastructure by deleting everything**

  terraform destroy

Exercise 02
Full Node

Exercise 02: Full Node

- **Introduces**: DigitalOcean volume, full node
- **Goal**: deploy a full Ethereum node on a DO droplet (w/data volume) using Terraform and Ansible
- **Terraform config**:
  - digitalocean_droplet parity_full
  - digitalocean_volume parity_data

Exercise 02: [1/4] build the infrastructure

- **Enter the exercise directory**

  ```
  cd 02_full_node
  ```

- **Initialize the Terraform configuration**

  ```
  terraform init
  ```

- **View the Terraform execution plan**

  ```
  terraform plan
  ```

- **Apply the Terraform config to build the infrastructure**

  ```
  terraform apply
  ```

Exercise 02: [2/4] run Ansible playbook

- **Run Ansible playbook**

  ```
  ansible-playbook -i terraform-inventory site.yml
  ```

Exercise 02: [3/4] query the node

Query the node using the JSON-RPC API

```
ip=$(terraform-inventory -list | jq -r .parity_node[0])
header="Content-Type: application/json”
host="http://ip:8545"
queries=“web3_clientVersion net_version net_peerCount”

for q in queries; do
  data="{'jsonrpc':'2.0','method':'$q','params':[],'id':67}"
  curl -H "$header" -X POST -data “$data”
done
```

Exercise 02: [4/4] clean up

- **Clean up the infrastructure by deleting everything**

  terraform destroy

Exercise 03
Reverse Proxy

Exercise 03: Reverse Proxy

- Demonstrates load balancing & high availability
- Simple setup with two backend web servers
- Web servers simply display a message with the hostname

Exercise 03: [1/4] build the infrastructure

- **Enter the exercise directory**

  ```
  cd 03_proxy
  ```

- **Initialize the Terraform configuration**

  ```
  terraform init
  ```

- **View the Terraform execution plan**

  ```
  terraform plan
  ```

- **Apply the Terraform config to build the infrastructure**

  ```
  terraform apply
  ```

Exercise 03: [2/4] run Ansible playbook

- **Run Ansible playbook**

  ```
  ansible-playbook -i terraform-inventory site.yml
  ```

Exercise 02: [3/4] observe running infrastructure

- **Observe the responses from multiple requests**

```
ip=$(terraform-inventory -list | jq -r .haproxy[0])
for i in `seq 1 10`; do curl -k $ip; sleep 1; done
```

- **View the HAProxy statistics page in web browser**

```
echo http://$ip/haproxy?stats
```

Exercise 03: [4/4] clean up

- **Clean up the infrastructure by deleting everything**

  terraform destroy

Exercise 04
Ethereum Application

Exercise 04: Ethereum Application

- Combines previous exercises into full application stack
  - Two Parity Ethereum full nodes behind HAProxy
- EthStats
  - App Server running eth-netstats
  - Parity nodes running eth-net-intelligence-api

Exercise 04: [1/5] build the infrastructure

- **Enter the exercise directory**

  ```
  cd 04_ethereum_app
  ```

- **Initialize the Terraform configuration**

  ```
  terraform init
  ```

- **View the Terraform execution plan**

  ```
  terraform plan
  ```

- **Apply the Terraform config to build the infrastructure**

  ```
  terraform apply
  ```

Exercise 04: [2/5] run Ansible playbook

- **Run Ansible playbook**

  ```
  ansible-playbook -i terraform-inventory site.yml
  ```

Exercise 04: [3/5] observe running infrastructure

- **View the EthStats web UI**

```
ip=$(terraform-inventory -list | jq -r .app_node[0])
echo http://$ip:3000
```

- **View HAProxy statistics web page**

```
ip=$(terraform-inventory -list | jq -r .haproxy[0])
echo http://$ip:8545/haproxy?stats
```

Exercise 04: [4/5] make JSON-RPC requests to proxy

Define convenience function for web3 queries

```
function web3query() {
  curl -s -H "Content-Type: application/json" \
    -X POST --data \
   "{\"jsonrpc\":\"2.0\",\"method\":\"$1\",\"params\":[],\"id\":74}" \
    http://$ip:8545 | jq -r .result
}
```

Send JSON-RPC requests to the proxy

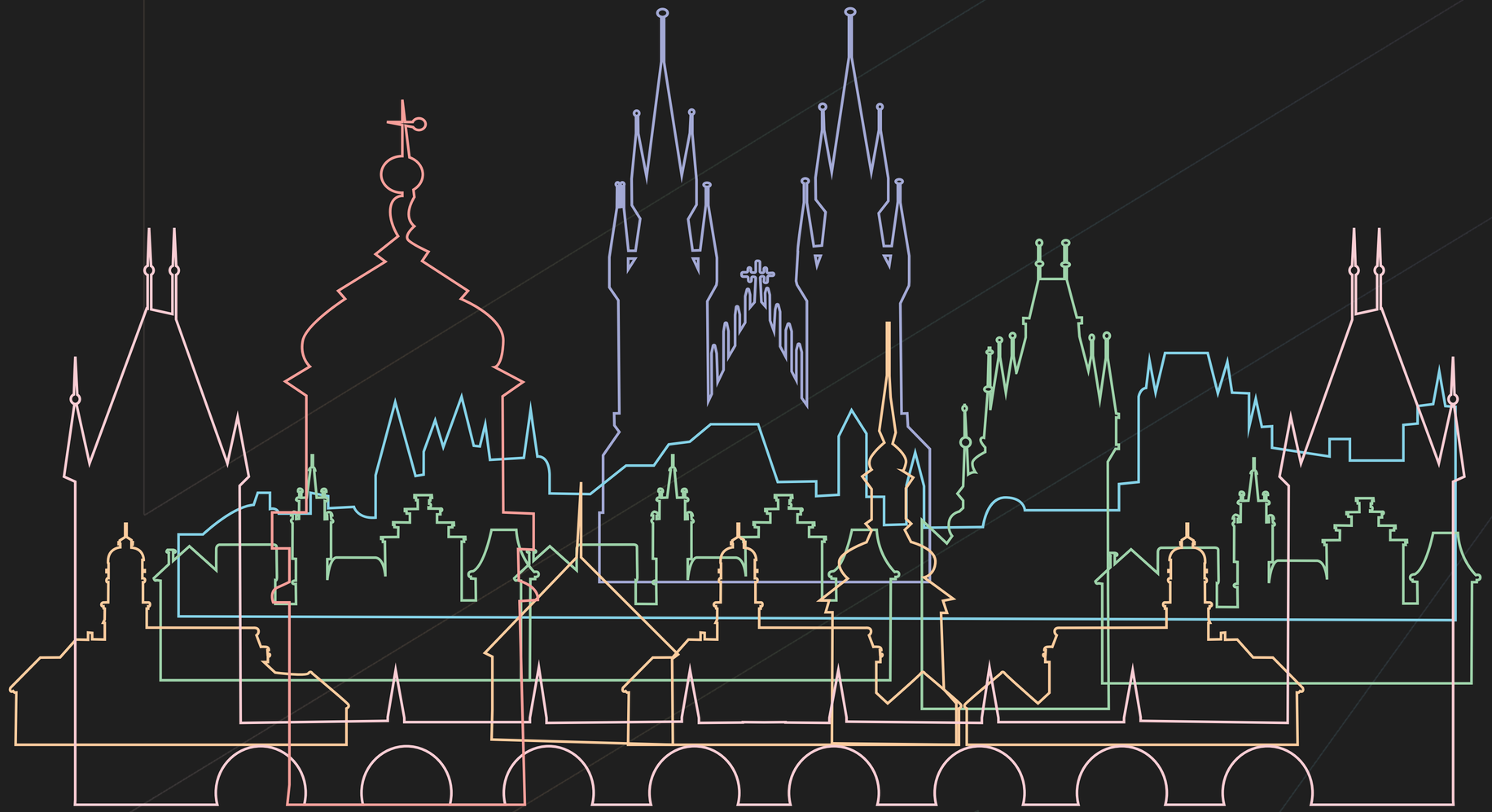
```
web3query web3_clientVersion
web3query net_version
web3query net_peerCount
```

Exercise 04: [5/5] clean up

- **Clean up the infrastructure by deleting everything**

  terraform destroy

Thank you for attending!



Follow the latest updates: https://www.CryptoKube.io

alex@thinkmassive.org | @thinkmassive | Denver, Colorado, USA