CITA-Cloud

Rivtower

2021 年 12 月 09 日

Contents

CITA-Cloud 是一个基于云原生的联盟链框架，用户可以基于该框架快速定制出一个适合具体业务场景的链。其主要特性有：

- **语言无关**。 整个框架采用微服务架构，不同微服务可以采用不同的编程语言。各种语言的开发者都可以方便的参与。也可以在开发过程中选择最适合的语言，方便复用已有的软件栈或者库。
- **组件解耦**。 微服务之间高度解耦，每个微服务可以有多种不同的实现，相互之间可以随意替换。可以形成组件市场。用户根据需要选择适合的组件，组成一条自定义的链。场景变化时，也可以快速切换其他的组件。
- **场景定制**。 如果已有的组件都不能满足用户需求，可以针对场景定制某个组件。同时可以复用其他组件，达到快速定制的目标。
- **兼容**。 得益于灵活的架构，可以复用已有区块链产品的组件，达到兼容其其生态的目的。也可以通过跨链的方式与其他区块链系统协作。
- Github 主页
- 官方网站
- 技术论坛
- rfcs

CHAPTER 1

CITA-Cloud 介绍

1.1 产品定位

参见产品定位白皮书。

1.2 运行环境

CITA-Cloud 默认运行在k8s环境中，原因有：

- 方便支持各种基础设施，尤其是各种云计算设施。
- 方便实施自动化运维。
- 方便复用云原生社区已有的技术。

1.3 工程仓库

CITA-Cloud 的代码仓库都在 github 上cita-cloud 组织下。主要代码仓库有：

- rfcs 存放白皮书以及后续的改进建议。
- cita_cloud_proto 存放 CITA-Cloud 微服务间 gRPC 接口定义文件。
- cita_cloud_config 用于生成链的配置文件的工具。
- charts 用于在 k8s 环境部署链的工程。

- docs 存放文档。

其余仓库主要是各个微服务实现。仓库名称以 `cita_cloud_proto` 中定义的微服务名称开头，下划线后接用于标识不同实现的名称。比如 `storage_sqlite`，是基于 `sqlite` 实现的 `storage` 微服务。发布件以 `Docker` 镜像的方式发布在 `DockerHub` 上，参见链接。

CHAPTER 2

快速入门

本章介绍的是使用 minikube 的快速入门操作方法，CITA-Cloud 提供本地运行版本 runner_local，参阅本地运行一节

2.1 环境准备

2.1.1 硬件配置建议

- CPU：2 核或以上
- 内存：4GB 或以上
- 硬盘：30G 或以上

2.1.2 操作系统

常见的 `Linux` 发行版本均可，例如：`CentOS`，`Debian`，`Ubuntu` 等等。

2.1.3 软件依赖

- docker 安装方法参见官方文档。
- helm 安装方法参见helm 文档。Helm 是 Kubernetes 的包管理器。Helm 是寻找、共享和使用为 Kubernetes 构建的软件的最佳方式。

2.1.4 k8s 集群

k8s 集群的搭建非常复杂，在快速入门中，我们推荐使用 minikube，可以在本地快速搭建一个单节点的 k8s 集群。

- minikube 安装方法参见官方文档。

安装完成后用下面的命令启动 minikube，国内需要在启动 minikube 时设置一些镜像参数，注意不能在根权限下启动 minikube。

```
minikube start --registry-mirror=https://hub-mirror.c.163.com --image-
↪repository=registry.cn-hangzhou.aliyuncs.com/google_containers --vm-driver=docker --
↪alsologtostderr -v=8 --base-image registry.cn-hangzhou.aliyuncs.com/google_
↪containers/kicbase:v0.0.17
```

耐心等待，看到以下信息代表启动成功。

```
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace␣
↪by default
```

- k8s 集群命令行工具 kubectl 安装方法参见官方文档。kubectl 是 Kubernetes 集群的命令行工具, 通过 kubectl 能够对集群本身进行管理, 并能够在集群上进行容器化应用的安装部署。

2.1.5 cloud-cli

该工具为 CITA-Cloud 链的命令行客户端，可以方便的对链进行常用的操作。

```
$ wget https://github.com/cita-cloud/cloud-cli/releases/download/v0.1.1/cldi
$ chmod +x cldi
$ sudo mv ./cldi /usr/local/bin/
$ cldi -h
cloud-cli 0.1.0
The command line interface to interact with `CITA-Cloud`.

USAGE:
    cldi [OPTIONS] [SUBCOMMAND]

FLAGS:
```

(下页继续)

(续上页)

```
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -e, --executor_addr <executor_addr>    executor address
    -r, --rpc_addr <rpc_addr>              rpc(controller) address
    -u, --user <user>                      the user(account) to send tx

SUBCOMMANDS:
    account               Manage account
    bench                 Send multiple txs with random content
    block-hash            Get block hash by block number(height)
    block-number          Get block number
    call                  Executor call
    completions           Generate completions for current shell
    create                Create contract
    emergency-brake       Send emergency brake cmd to chain
    get-abi               Get specific contract abi
    get-balance           Get balance by account address
    get-block             Get block by number or hash
    get-code              Get code by contract address
    get-tx                Get transaction by hash
    help                  Prints this message or the help of the given subcommand(s)
    peer-count            Get peer count
    receipt               Get receipt by tx_hash
    send                  Send transaction
    set-block-interval    Set block interval
    store-abi             Store abi
    system-config         Get system config
    update-admin          Update admin of the chain
    update-validators     Update validators of the chain
```

2.2 运行 CITA-Cloud

2.2.1 添加 Charts 仓库

```
$ helm repo add cita-cloud https://registry.devops.rivtower.com/chartrepo/cita-cloud
$ helm repo update
$ helm search repo cita-cloud
NAME                                            CHART VERSION   APP VERSION     ␣
↪DESCRIPTION
```

(下页继续)

(续上页)

```
cita-cloud/cita-cloud-config                    6.3.0           6.3.0           ␣
↪Create a job to change config of CITA-Cloud blo...
cita-cloud/cita-cloud-eip                       6.3.0           6.3.0           ␣
↪Create EIP for CITA-Cloud
cita-cloud/cita-cloud-local-cluster             6.3.0           6.3.0            Setup␣
↪CITA-Cloud blockchain in one k8s cluster
cita-cloud/cita-cloud-multi-cluster-node        6.3.0           6.3.0            Setup␣
↪CITA-Cloud node in multi k8s cluster
cita-cloud/cita-cloud-porter-lb                 6.3.0           6.3.0            Setup␣
↪porter Loadbalancer for CITA-Cloud node
cita-cloud/cita-cloud-pvc                       6.0.0           6.0.0           ␣
↪Create PVC for CITA-Cloud
```

2.2.2 创建 PVC

PersistentVolumeClaim (PVC) 是对 PV 的申请 (Claim)。PVC 通常由普通用户创建和维护。需要为 Pod 分配存储资源时，用户可以创建一个 PVC, 指明存储资源的容量大小和访问模式 (比如只读) 等信息，Kubemetes 会查找并提供满足条件的 PV。

```
$ helm install local-pvc cita-cloud/cita-cloud-pvc --set scName=standard
$ kubectl get pvc
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS␣
↪MODES   STORAGECLASS    AGE
local-pvc   Bound    pvc-fd3eaebd-3413-4205-b88a-dbc6cee9a057   10Gi       RWO       ␣
↪    standard        18m
```

对应的路径在 `minikube` 虚拟机内的 `/tmp/hostpath-provisioner/default/local-pvc`。

注意：如果 `minikube` 版本为 `v1.20.0`，这里会有一个 bug。详细情况和解决方法参见链接。

2.2.3 生成超级管理员账户

```
$ cldi account create admin
user: `admin`
account_addr: 0xae069e1925a1dad2a1f4c7034d87258dfd9b6532
```

2.2.4 运行 CITA-Cloud

```
$ helm install test-chain cita-cloud/cita-cloud-local-cluster --set config.
↪superAdmin=0xae069e1925a1dad2a1f4c7034d87258dfd9b6532
NAME: test-chain
LAST DEPLOYED: Wed Jul 14 23:09:37 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
```

该命令会创建一条有 3 个节点，链名为 `test-chain` 的链。

注意：`superAdmin` 参数必须设置为自己生成的账户地址，切勿使用默认值。

2.2.5 查看运行情况

```
$ kubectl get pod
NAME                          READY       STATUS        RESTARTS      AGE
test-chain-0                  7/7         Running       1             3m24s
test-chain-1                  7/7         Running       1             3m24s
test-chain-2                  7/7         Running       1             3m24s
```

查看日志：

```
$ minikube ssh
docker@minikube:~$ tail -10f /tmp/hostpath-provisioner/default/local-pvc/test-chain-0/
↪logs/controller-service.log
2021-07-21T08:12:07.242086786+00:00 INFO controller::node_manager - update node:␣
↪0xc3469...ebaeb
2021-07-21T08:12:07.242251607+00:00 INFO controller::node_manager - update node:␣
↪0xa6a1...208f9
2021-07-21T08:12:07.284207745+00:00 INFO controller::controller - add remote␣
↪proposal(0x7b28...5cc64) through check_proposal
2021-07-21T08:12:10.228046821+00:00 INFO controller::controller - add remote␣
↪proposal(0x26e7...fcf91) through check_proposal
2021-07-21T08:12:10.251289942+00:00 INFO controller::util - height: 126 hash 0x26e7...
↪fcf91
```

(下页继续)

(续上页)

```
2021-07-21T08:12:10.253093372+00:00 INFO controller::chain - finalize_block: 126,␣
↪block_hash: 0x26e7...fcf91
2021-07-21T08:12:10.254012416+00:00 INFO controller::node_manager - update node:␣
↪0xc346...ebaeb
2021-07-21T08:12:10.254931221+00:00 INFO controller::node_manager - update node:␣
↪0x766a...7a954
2021-07-21T08:12:10.273834898+00:00 WARN controller - rpc: process_network_msg␣
↪failed: Receive early status from same node
2021-07-21T08:12:10.296086993+00:00 INFO controller::controller - add remote␣
↪proposal(0xe5f4...973cd) through check_proposal
2021-07-21T08:12:13.216728462+00:00 INFO controller::chain - main_chain_tx_hash len 0
2021-07-21T08:12:13.216755352+00:00 INFO controller::chain - tx poll len 0
2021-07-21T08:12:13.216766922+00:00 INFO controller::chain - proposal 127 prevhash␣
↪0x26e7...fcf91
2021-07-21T08:12:13.216778762+00:00 INFO controller::chain - proposal 127 block_hash␣
↪0x6a14...1e4e6
2021-07-21T08:12:13.256383088+00:00 INFO controller::util - height: 127 hash 0x6a14...
↪1e4e6
```

2.3 基本操作

2.3.1 指定链的 RPC 端口

可以通过设置环境变量的方式，为 `cloud-cli` 工具指定链的 `RPC` 端口：

```
$ export CITA_CLOUD_RPC_ADDR=`minikube ip`:30004 CITA_CLOUD_EXECUTOR_ADDR=`minikube␣
↪ip`:30005
```

注意：这里 minikube 可能出现 svc 端口映射问题使用以下命令解决

```
$ kubectl port-forward pod/test-chain-0 50002:50002 50004:50004
$ export CITA_CLOUD_RPC_ADDR=localhost:30004 CITA_CLOUD_EXECUTOR_ADDR=localhost:30005
```

2.3.2 查看块高

```
$ cldi block-number
block_number: 74
```

2.3.3 查看系统配置

```
$ cldi system-config
{
  "admin": "0xae069e1925a1dad2a1f4c7034d87258dfd9b6532",
  "block_interval": 3,
  "chain_id": "0x26b0b83e7281be3b117658b6f2636d0368cad3d74f22243428f5401a4b70897e",
  "validators": [
    "0x67611c4afee6a50c56d3a81c733260c2e1ca35ab",
    "0x97e05af22f5870c67b7f98bc6c7ebbba0273376b",
    "0x3275a8e90a92a29496edd9a7a5853a3fd3d51451",
    "0x148d37bd89ba2a8c7b5e4784df51a355439166b9"
  ],
  "version": 0
}
```

2.3.4 停止链

```
$ helm uninstall test-chain
release "test-chain" uninstalled
```

2.3.5 删除链

```
$ helm install clean cita-cloud/cita-cloud-config --set config.action.type=clean
NAME: clean
LAST DEPLOYED: Wed Jul 14 20:20:37 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

注意：该命令将永久性的删除链的所有数据，请谨慎操作。

2.4 账户操作

2.4.1 创建账户

```
$ cldi account create test
user: `test`
account_addr: 0x415f568207900b6940477396fcd2c201efe49beb
```

2.4.2 查看账户信息

创建账户后通过命令查看该账户的账户地址、私钥以及公钥:

```
$ cldi account export test
{
  "account_addr": "0x415f568207900b6940477396fcd2c201efe49beb",
  "private_key": "0x4f894fc00e6c71c7d0dde511eef64824b64fce87fd6f43492808cb99e9f22a57",
  "public_key":
↪"0x6dd968546f2af3053be1d31aed723b58bf5380884ec5f2d41e8156dcc17c1317456c2cc9fb28290d7da0e606267ec1b0
↪"
}
```

2.4.3 登录账户

登录刚刚创建的 `test` 账户

```
$ cldi account login test
OK, now the default user is `test`, account addr is␣
↪0x415f568207900b6940477396fcd2c201efe49beb
```

登录账户之后，可以以该账户的身份发送交易。

2.5 发送交易

2.5.1 编译合约

这里以 `Counter` 合约为例:

```
$ cat Counter.sol
pragma solidity ^0.4.24;

contract Counter {
    uint public count;

    function add() public {
```

(下页继续)

(续上页)

```
        count += 1;
    }

    function reset() public {
        count = 0;
    }
}

$ curl -o solc -L https://github.com/ethereum/solidity/releases/download/v0.4.24/solc-
↪static-linux
$ chmod +x solc
$ sudo mv ./solc /usr/local/bin/
$ solc --hashes --bin Counter.sol

======= Counter.sol:Counter =======
Binary:
608060405234801561001057600080fd5b5060f58061001f6000396000f3006080604052600436106053576000357c0100000
Function signatures:
4f2be91f: add()
06661abd: count()
d826f88f: reset()
```

2.5.2 创建合约

```
$ cldi create␣
↪0x608060405234801561001057600080fd5b5060f58061001f6000396000f3006080604052600436106053576000357c010
tx_hash: 0xb8fdb0a82d0403921fd0c11c7846500bdb5c032ad2e615dd1e8f42439f38f516
```

创建合约的参数是编译合约输出的二进制字节码，注意前面要增加 `0x` 前缀。

返回值为这笔创建合约交易的交易哈希。

2.5.3 查看交易回执

```
$ cldi receipt 0xb8fdb0a82d0403921fd0c11c7846500bdb5c032ad2e615dd1e8f42439f38f516
{
  "block_number": 1111,
  "contract_addr": "0x138e2c0098ab9a5c38a7bc4a16a186f58fc4058a",
  "cumulative_quota_used":
↪"0x00000000000000000000000000000000000000000000000000000000000018ed3",
  "error_msg": "",
```

(下页继续)

(续上页)


```
  "legacy_cita_block_hash":
↪"0x1b99b5e81feaef86dca9a309644ddbaaa7377733712b850ce50aff6c01ab2d4e",
  "logs": [],
  "logs_bloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
  "quota_used": "0x000000000000000000000000000000000000000000000000000000000018ed3",
  "state_root": "0x9584cba462dec67dafaf5ae1210c8e3b3e58991ca31cafdbc395884b7d880364",
  "tx_hash": "0xb8fdb0a82d0403921fd0c11c7846500bdb5c032ad2e615dd1e8f42439f38f516",
  "tx_index": 0
}
```


参数为前一步操作返回的交易哈希。

返回值为该笔交易的执行结果信息，其中 `contract_addr` 字段为刚才部署的合约的地址。

得到合约地址后，就可以调用其中的方法。

查询合约状态

查询合约中的 `count` 值：

```
$ cldi call -t 0x138e2c0098ab9a5c38a7bc4a16a186f58fc4058a 0x06661abd
result: 0x0000000000000000000000000000000000000000000000000000000000000000
```

`-t` 参数为合约地址。

第二个参数为要调用的 `count()` 方法的函数签名。

返回值为 `count` 的当前值。

2.5.4 发送交易

向合约发送交易，调用 `add` 方法改变 `count` 的值：

```
$ cldi send -t 0x138e2c0098ab9a5c38a7bc4a16a186f58fc4058a 0x4f2be91f
```

`-t` 参数为合约地址。

第二个参数为要调用的 `add()` 方法的函数签名。

等待交易上链之后，再次查询就可以发现合约状态有了变化：

```
$ cldi call -t 0x138e2c0098ab9a5c38a7bc4a16a186f58fc4058a 0x06661abd
result: 0x0000000000000000000000000000000000000000000000000000000000000001
```

CHAPTER 3

本地运行

runner_local是 CITA-Cloud 的本地运行版本，可以方便不熟悉 k8s 运行环境的用户在本地体验 CITA-Cloud。

3.1 获取工程及子模块文件

- 首先使用 `git clone` 命令获取 `runner_local` 文件，命令如下：

```
$ git clone git@github.com:cita-cloud/runner_local.git
```

- 使用 `git checkout` 切换到需要使用的版本分支。
- 适用以下命令获取 runner_local 中的各个子模块

```
$ git submodule update --init --remote
```

3.2 生成配置

使用cloud-config生成链的配置文件。

- 安装依赖

```
$ curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

$ apt install -y --no-install-recommends make git protobuf-compiler libssl-dev pkg-
↪config clang
```

(下页继续)

(续上页)

- 安装 cloud-config

```
$ cargo install --path config
```

- 使用 cloud-config create-dev 命令生成配置

```
$ cloud-config create-dev --help
cloud-config-create-dev

create config in env dev

USAGE:
    cloud-config create-dev [FLAGS] [OPTIONS]

FLAGS:
    -h, --help       Print help information
        --is-bft     is consensus bft
        --is-tls     is network tls
    -V, --version    Print version information

OPTIONS:
        --chain-name <CHAIN_NAME>      set chain name [default: test-chain]
        --config-dir <CONFIG_DIR>      set config file directory, default means␣
↪current directory[default: .]
        --log-level <LOG_LEVEL>        log level [default: info]
        --peers-count <PEERS_COUNT>    set initial node number [default: 4]
```

```
$ cloud-config create-dev --is-bft --config-dir tmp
```

- 查看生成的配置

```
$ tree tmp/ -L 1
tmp
├── test-chain
├── test-chain-0
├── test-chain-1
├── test-chain-2
└── test-chain-3

5 directories, 0 files
```

3.3 编译和使用

编译工程

```
$ make release
```

编译的二进制和配置文件会放到 target/install 目录下，编译完成后切换至该目录。

```
$ cd target/install
```

注意

在使用前请先参阅快速入门-环境准备-cloud-cli，安装 CITA-Cloud 命令行工具 cloud-cli，可以方便地对链进行常用操作。

- 启动链

```
$ ./scripts/env.sh start config/test-chain-0 50000 && ./scripts/env.sh start config/
↪test-chain-1 51000 && ./scripts/env.sh start config/test-chain-2 52000 && ./scripts/
↪env.sh start config/test-chain-3 53000
```

启动成功后就可以使用 cloud-cli 与链进行交互了，链的操作与使用 k8s 运行的 CITA-Cloud 一样，参阅快速入门中基本操作、账户操作、发送交易三节。

- 停止链

```
$ ./scripts/env.sh stop
```

- 删除链文件

```
$ ./scripts/env.sh clean config/test-chain-0 && ./scripts/env.sh clean config/test-
↪chain-1 && ./scripts/env.sh clean config/test-chain-2 && ./scripts/env.sh clean␣
↪config/test-chain-3
```

CHAPTER 4

配置说明

区块链是一个分布式对等网络，但是其配置需要集中生成。在 CITA-Cloud 中每个节点都要包含其他节点的信息，因此需要将所有节点的信息集中到一起，生成适用于各个节点的配置文件，然后再下发给各个节点分别运行。

CITA-Cloud 的配置生成工具为cloud-config。

> **注意**
>
> 当你计划使用 CITA-Cloud 设计产品时，[环境准备] 好之后，不要着急启动节点，请仔细阅读本节内容，并选择最适合你产品需求的配置。

本文档会详细介绍链的各个可配置项。

4.1 链级配置

链级配置指的是链自身的一些属性，系统初始配置、创世块、节点网络地址等配置，用户需在起链前初始化链级配置，以下是生成链级配置的相关命令。

4.1.1 init-chain 命令

根据指定的 config-dir 和 chan-name, 初始化一个链的文件目录结构。

参数:

```
--chain-name <CHAIN_NAME>
        set chain name [default: test-chain]
--config-dir <CONFIG_DIR>
        set config file directory, default means current directory [default: .]
```

4.1.2 init-chain-config 命令

初始化除 admin(管理员账户)，validators(共识节点地址列表)，node_network_address_list（节点网络地址列表）之外的链级配置。因为前述三个操作需要一些额外的准备工作，且需要先对除此之外的链接配置信息在所有参与方之间达成共识。因此对于去中心化场景来说，这一步其实是一个公示的过程。执行之后会生成 $(config-dir)/$(chain-name)/chain_config.toml。该命令有以下参数:

--block_interval

设置出块时间间隔，默认值为 3。

--block_limit

设置内存中存储区块的上限，默认值为 100，即内存中存储最近 100 个区块。

--chain_name

设置链的名字。

- 链的名字会作为文件夹的名称。以 test-chain 为例，按节点序号分别创建 test-chain-0，test-chain-1，test-chain-2 等节点文件夹，分别存放每个节点的配置文件。
- 如果没有传递 chain_name 参数，则默认链的名字为 test-chain。

--chain_id

设置链 id，默认为空字符串。检测到为默认值时，自动替换为 hex(sm3($(chain_name)))。

--config-dir

设置配置文件目录，默认为当前目录。

--timestamp

设置起链的时间戳，默认参数为 0。检测到为默认值时，自动替换为当前时间对应的时间戳。

- 具体数值是指自 `1970-1-1` 以来的毫秒数，默认是取当前的时间。

--prevhash

设置创世块的父哈希值，默认为 0x0000000000000000000000000000000000000000000000000000000000000000

--version

设置配置版本信息。

微服务选择相关参数

使用 `init-chain-config` 命令设置微服务的相关参数如下

```
--consensus_image <CONSENSUS_IMAGE>
        set consensus micro service image name (consensus_bft/consensus_raft)␣
↪[default:consensus_raft]

--consensus_tag <CONSENSUS_TAG>
        set consensus micro service image tag [default: latest]

--controller_image <CONTROLLER_IMAGE>
        set controller micro service image name (controller)[default:controller]

--controller_tag <CONTROLLER_TAG>
        set controller micro service image tag [default: latest]

--executor_image <EXECUTOR_IMAGE>
        set executor micro service image name (executor_evm) [default: executor_evm]

--executor_tag <EXECUTOR_TAG>
```

(下页继续)

(续上页)

```
        set executor micro service image tag [default: latest]

--kms_image <KMS_IMAGE>
        set kms micro service image name (kms_eth/kms_sm) [default: kms_sm]

--kms_tag <KMS_TAG>
        set kms micro service image tag [default: latest]

--network_image <NETWORK_IMAGE>
        set network micro service image name (network_tls/network_p2p) [default:␣
↪network_p2p]

--network_tag <NETWORK_TAG>
        set network micro service image tag [default: latest]

--storage_image <STORAGE_IMAGE>
        set storage micro service image name (storage_rocksdb) [default: storage_
↪rocksdb]

--storage_tag <STORAGE_TAG>
        set storage micro service image tag [default: latest]
```

4.1.3 set-admin 命令

设置管理员账户。账户需要事先通过 new-account 子命令（见辅助命令一节）创建。如果网络微服务选择了 network_tls，则还需要通过 create-ca 创建链的根证书。

参数：

```
--admin <ADMIN>
        set admin
--chain-name <CHAIN_NAME>
        set chain name [default: test-chain]
--config-dir <CONFIG_DIR>
        set config file directory,default means currentdirectory[default:.]
```

- admin 为必选参数。值为之前用 new-account 创建的地址。

4.1.4 set-validators 命令

设置共识节点账户列表。账户同样需要事先通过 new-account 子命令（见辅助命令一节），由各个共识节点分别创建，然后将账户地址集中到一起进行设置。

参数:

```
--chain-name <CHAIN_NAME>
        set chain name [default: test-chain]
--config-dir <CONFIG_DIR>
        set config file directory,default means current directory[default:.]
--validators <VALIDATORS>
        validators account splited by ','
```

- validators 为必选参数。值为多个之前用 new-account 创建的地址，用逗号分隔。

4.1.5 set-nodelist 命令

设置节点网络地址列表。各个节点参与方需要根据自己的网络环境，预先保留节点的 ip，port 和 domain。然后将相关信息集中到一起进行设置。至此，链级配置信息设置完成，可以下发配置文件 chain_config.toml 到各个节点。如果网络微服务选择了 network_tls，则需要通过 create-csr 根据节点的 domain 为各个节点创建证书和签名请求。然后请求 CA 通过 sign-crs 处理签名请求，并下发生成的 cert.pem 到各个节点。

参数:

```
--chain-name <CHAIN_NAME>
        set chain name [default: test-chain]
--config-dir <CONFIG_DIR>
        set config file directory,default means current directory[default:.]
--nodelist <NODE_LIST>
        node list looks like localhost:40000:node0,localhost:40001:node1
```

- nodelist 为必选参数。值为多个节点的网络地址，用逗号分隔。每个节点的网络地址包含 ip,port 和 domain，之间用冒号分隔。
- domain 为任意字符串，只需要确保节点之间不重复即可。

4.2 辅助命令

4.2.1 create-ca 命令

创建链的根证书。会在 `$(config-dir)/$(chain-name)/ca_cert/`下生成 `cert.pem` 和 `key.pem` 两个文件。

参数:

```
--chain-name <CHAIN_NAME>
        set chain name [default: test-chain]
--config-dir <CONFIG_DIR>
        set config file directory, default means current directory [default: .]
```

`--chain-name` 设置链的名称，默认为 test-chain。`--config-dir` 设置配置文件目录，默认为当前目录。

4.2.2 create-csr 命令

为各个节点创建证书和签名请求。会在 `$(config-dir)/$(chain-name)/certs/$(domain)/`下生成 `csr.pem` 和 `key.pem` 两个文件。

参数:

```
--chain-name <CHAIN_NAME>
        set chain name [default: test-chain]
--config-dir <CONFIG_DIR>
        set config file directory, default means current directory [default: .]
--domain <DOMAIN>
        domain of node
```

`--chain-name` 设置链的名称，默认为 test-chain。`--config-dir` 设置配置文件目录，默认为当前目录。`--domain` 为必选参数。值为前面 `set-nodelist` 或者 `append-node` 时传递的节点的网络地址中的 `domain`。

4.2.3 sign-csr 命令

处理节点的签名请求。会在 `$(config-dir)/$(chain-name)/certs/$(domain)/`下生成 `cert.pem`。

参数:

```
--chain-name <CHAIN_NAME>
        set chain name [default: test-chain]
--config-dir <CONFIG_DIR>
```

(下页继续)

(续上页)

```
        set config file directory, default means current directory [default: .]
--domain <DOMAIN>
        domain of node
```

--chain-name 设置链的名称，默认为 test-chain。--config-dir 设置配置文件目录，默认为当前目录。domain 为必选参数。值为前面执行 create-csr 时节点的 domain。

4.2.4 new-account 命令

创建账户。会在 $(config-dir)/$(chain-name)/accounts/下，创建以账户地址为名的文件夹，里面有 key_id 和 kms.db 两个文件。

参数：

```
--chain-name <CHAIN_NAME>
        set chain name [default: test-chain]
--config-dir <CONFIG_DIR>
        set config file directory, default means current directory [default: .]
--kms-password <KMS_PASSWORD>
        kms db password [default: 123456]
```

--chain-name 设置链的名称，默认为 test-chain。--config-dir 设置配置文件目录，默认为当前目录。--kms-password 输入密钥库密码，默认为 “123456”。

4.3 节点配置

节点配置指与链级配置无关的单个节点内部的配置，

4.3.1 init-node 命令

设置节点配置信息。这步操作由各个节点的参与方独立设置，节点之间可以不同。执行之后会生成 $(config-dir)/$(chain-name)-$(domain)/node_config.toml。有以下参数：

--account

account 为必选参数，表示该节点要使用的账户地址。值为之前用 new-account 创建的地址。

--chain-name

设置链的名字，默认为 `test-chain`，若生成链级配置时没有采用默认值这里也要对应。

--config-dir

设置配置文件目录，默认为当前文件夹，若生成链级配置时没有采用默认值这里也要对应。

--key-id

密钥库中的账户密钥 id，默认为 1。

--kms-password

密钥库密码，默认为”123456”。

--log-level

生成日志的等级，默认为 info。

--package-limit

单个区块中包含交易量的上限，默认为 30000。

--network-listen-port

本节点供区块链网络中其他节点连接的端口，默认为 40000。

微服务 grpc 端口参数

通过以下参数设置节点内部各个微服务之间通信使用的 `grpc` 端口。

```
--network-port <NETWORK_PORT>
        grpc network_port of node [default: 50000]
--consensus-port <CONSENSUS_PORT>
        grpc consensus_port of node [default: 50001]
        --executor-port <EXECUTOR_PORT>
        grpc executor_port of node [default: 50002]
        --storage-port <STORAGE_PORT>
        grpc storage_port of node [default: 50003]
--controller-port <CONTROLLER_PORT>
```

(下页继续)

(续上页)

```
        grpc controller_port of node [default: 50004]
--kms-port <KMS_PORT>
        grpc kms_port of node [default: 50005]
```

4.3.2 update-node 命令

根据之前设置的链级配置和节点配置，生成每个节点所需的微服务配置文件。

--chain-name

设置链的名字，默认为 test-chain，若生成链级配置时没有采用默认值这里也要对应。

--config-dir

设置配置文件目录，默认为当前文件夹，若生成链级配置时没有采用默认值这里也要对应。

--config-name

设置节点配置信息源，默认为 config.toml，即保存 init-node 命令中生成的配置信息的文件。

--domain

domain 为必选参数，作为节点的标识，表示要操作的节点。

4.4 各微服务内部参数

在生成配置时各个微服务中一些参数可能会采用默认值，这些参数在配置生成后也可以在各节点的 config.toml 文件中手动修改，以下是以各微服务为划分，关于这些参数的说明。

4.4.1 network_p2p

```
[network_p2p]
grpc_port = 50000
port = 40000

[[network_p2p.peers]]
address = '/dns4/127.0.0.1/tcp/40001'
```

(下页继续)

(续上页)

```
[[network_p2p.peers]]
address = '/dns4/127.0.0.1/tcp/40002'

[[network_p2p.peers]]
address = '/dns4/127.0.0.1/tcp/40003'
```

说明:

1. `[network_p2p]` 中是本节点的网络配置信息，`grpc_port` 是本节点网络微服务和其他微服务通信的 grpc 端口，`port` 是本节点供区块链网络中其他节点连接的端口。
2. `[network_p2p.peers]` 中是本节点连接的其他网络节点的信息，以 multiaddr 标准记录，`/dns4` 后是 host 信息，`/tcp` 后是 port 信息。

4.4.2 network_tls

```
[network_tls]
grpc_port = 50000
listen_port = 40000
reconnect_timeout = 5

[[network_tls.peers]]
domain = 'test-chain-1'
host = '127.0.0.1'
port = 40001

[[network_tls.peers]]
domain = 'test-chain-2'
host = '127.0.0.1'
port = 40002

[[network_tls.peers]]
domain = 'test-chain-3'
host = '127.0.0.1'
port = 40003
```

说明:

1. `[network_tls]` 中是本节点的网络配置信息，`grpc_port` 是本节点网络微服务和其他微服务通信的 grpc 端口，`listen_port` 是本节点供区块链网络中其他节点连接的端口，`reconnect_timeout` 是超时重连时间。
2. `[network_tls.peers]` 中是本节点连接的其他网络节点的信息，其中 `domain` 为任意字符串，只需要确保节点之间不重复即可。。

4.4.3 consensus_raft

```
[consensus_raft]
controller_port = 50004
grpc_listen_port = 50001
network_port = 50000
node_addr = 'c7e1fe8c89790ef0f4c0548e759c849806475a48'
```

说明:

grpc_listen_port 是本节点共识微服务和其他微服务通信的 grpc 端口，controller_port 是共识微服务的 gRPC 端口，network_port 是网络微服务的 gRPC 端口，node_addr 是本节点的地址。

4.4.4 consensus_bft

```
[consensus_bft]
consensus_port = 50001
controller_port = 50004
kms_port = 50005
network_port = 50000
node_address = '0x30bc783ff00ec6fb347a5b1c7b2480ae65dd007a'
```

说明:

consensus_port 是本节点共识微服务和其他微服务通信的 grpc 端口，controller_port 是共识微服务的 gRPC 端口，kms_port 是 kms 微服务的 gRPC 端口，network_port 是网络微服务的 gRPC 端口，node_addr 是本节点的的地址。

4.4.5 executor_evm

```
[executor_evm]
executor_port = 50002
```

说明:

executor_port 是执行器微服务的 gRPC 端口。

4.4.6 storage_rocksdb

```
[storage_rocksdb]
kms_port = 50005
storage_port = 50003
```

说明:

kms_port 是 kms 微服务的 gRPC 端口，storage_port 是存储微服务的 gRPC 端口。

4.4.7 controller

```
[controller]
consensus_port = 50001
controller_port = 50004
executor_port = 50002
key_id = 1
kms_port = 50005
network_port = 50000
node_address = 'c7e1fe8c89790ef0f4c0548e759c849806475a48'
package_limit = 30000
storage_port = 50003
```

说明:

consensus_port 是共识微服务的 gRPC 端口，controller_port 是控制器微服务的 gRPC 端口，executor_port 是执行器微服务的 gRPC 端口，kms_port 是 kms 微服务的 gRPC 端口，network_port 是网络微服务的 gRPC 端口，storage_port 是存储微服务的 gRPC 端口。

4.4.8 kms

```
db_key = '123456'
kms_port = 50005
```

说明:

kms_sm 和 kms_eth 相同，db_key 是密钥库的密码，kms_port 是 kms 微服务的 gRPC 端口。

CHAPTER 5

部署指南

部署一条 CITA-Cloud 链，除了准备好 k8s 集群外，还需要实现进行持久化存储和网络的设置。

5.1 持久化存储

链的节点是有状态的服务，需要挂载持久化存储来保存数据。

为了方便对接不同的存储服务，我们使用了 k8s 的 PV/PVC 对存储进行了抽象。

建议由运维人员配置 StorageClass，对 PV/PVC 进行动态绑定。

例如快速入门中就使用了 minikube 自带的名为 standard 的 StorageClass。

```
$ helm install local-pvc cita-cloud/cita-cloud-pvc --set scName=standard
```

用户需要根据自己的环境以及所使用的存储服务来设置 scName 参数。

local-pvc 为创建的 PVC 的名字，用户也可以随意设置。

5.2 网络

网络方面，需要节点之间可以通过网络相互连接。

集群内部可以通过 k8s 的 SVC 来暴露节点的网络端口。

如果是跨集群的情况，则需要使用 LoaderBalancer 服务对外暴露节点的网络端口。

如使用公有云环境，请咨询当前使用的云服务商。

5.3 部署模式

部署模式可分为单集群和多集群。

5.3.1 单集群

链的所有节点都在同一个 k8s 集群中。

我们提供了一个 Chart 工程来实现该部署模式。在此之前需先执行快速入门-运行 CITA-Cloud 中的添加 Charts 仓库和创建 PVC。

```
$ helm install test-chain cita-cloud/cita-cloud-local-cluster --set config.
↪superAdmin=0xae069e1925a1dad2a1f4c7034d87258dfd9b6532 --set pvcName=local-pvc
```

- test-chain 为要创建的链的名字。
- pvcName 参数指定了 PVC 的名字。
- superAdmin 要修改为自己的管理员地址。
- CITA-Cloud 的各个微服务都有多种实现，用户可以通过 xxx.imageName 和 xxx.imageTag 参数来选择要使用的实现。
- 更多参数参见链接。
- 部署上采用 statefulset，链的每个节点对应一个 pod。
- 网络使用了 headless service，使得节点直接可以相互访问。

5.3.2 多集群

链的节点分布在多个 k8s 集群中。

步骤如下：

1. 规划节点所属的 k8s 集群。
2. 提前设置好节点网络端口对外的 ip 和端口。
3. 提前设置好各 k8s 集群节点的 pvc。
4. 集中生成配置。
5. 将生成好的节点文件夹，分别下发到所属的 k8s 集群。
6. 在各个 k8s 集群中分别运行节点。

第 2 步对外暴露节点网络端口。

如果使用porterLB，可以使用如下命令创建 eip。

```
$ helm install cita-cloud-eip cita-cloud/cita-cloud-eip
```

- cita-cloud-eip 为 eip 的名字。
- 更多参数参见链接。

然后使用如下命令创建 SVC。

```
$ helm install test-chain-0-lb cita-cloud/cita-cloud-porter-lb
```

- test-chain-0-lb 为 SVC 的名字。
- 更多参数参见链接。

第 4 步生成配置。

可以使用 Chart 工程：

```
$ helm install init-multi cita-cloud/cita-cloud-config --set config.action.
↪type=initMulti --set config.chainName=test-chain --set config.action.initMulti.
↪superAdmin=8f81961f263f45f88230375623394c9301c033e7 --set config.action.initMulti.
↪kmsPasswordList="123456\,123456\,123456" --set config.action.initMulti.nodeList=
↪"192.168.10.123:40000:node0\,192.168.10.134:40000:node1\,192.168.10.135:40000:node2
↪" --set pvcName=local-pvc
```

更多参数参见链接。

也可以直接使用cita-cloud-config。

第 6 步在各个 k8s 运行节点。

可以使用 Chart 工程：

```
$ helm install test-chain-node0 cita-cloud/cita-cloud-multi-cluster-node --set config.
↪chainName=test-chain --set config.domain=node0
```

- chainName 要与第 4 步保持一致, domain 必须是第 4 步中其中一个节点的 domain。
- CITA-Cloud 的各个微服务都有多种实现，用户可以通过 xxx.imageName 和 xxx.imageTag 参数来选择要使用的实现。
- 更多参数参见链接。

针对阿里云场景，可以使用cita_cloud_operator。CITA-Cloud 的各个微服务都有多种实现，用户可以通过 service-config.toml 配置文件来选择要使用的实现。

CHAPTER 6

升级说明

CHAPTER 7

节点管理

7.1 节点分类

节点分为两大类：

- 共识节点：共识节点具有出块和投票权限，交易由共识节点排序并打包成块，共识完成后即被确认为合法区块。链创建时生成的节点默认为共识节点，后续管理员可修改共识节点成员。
- 普通节点：普通节点没有出块和投票权限，其他方面和共识节点相同。可以同步和验证链上所有的原始数据，接受交易数据并向其他节点广播。链创建之后添加的节点默认为普通节点。

公有链没有节点准入机制，意味着任何节点都可以接入链并同步其全部的数据，在满足一定的条件下都可以参加共识。而 `cita-cloud` 对于共识节点和普通节点都进行了准入管理。对于身份验证失败的节点，即使该节点能够在网络层与其他节点连通，这些节点也会拒绝与之建立通讯会话，如此可避免信息泄漏。

7.2 共识节点管理

参见治理-共识节点管理

7.3 普通节点管理

普通节点的管理，是指普通节点的添加与删除。

7.3.1 创建链

此时生成的三个节点默认是共识节点。

```
$ helm install init-multi cita-cloud/cita-cloud-config --set config.action.
↪type=initMulti --set config.chainName=test-chain --set config.action.initMulti.
↪superAdmin=8f81961f263f45f88230375623394c9301c033e7 --set config.action.initMulti.
↪kmsPasswordList="123456\,123456\,123456" --set config.action.initMulti.nodeList=
↪"192.168.10.123:40000:node0\,192.168.10.134:40000:node1\,192.168.10.135:40000:node2
↪" --set pvcName=local-pvc
NAME: init
LAST DEPLOYED: Thu Jul 29 23:29:52 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

生成文件

```
$ cd /tmp/hostpath-provisioner/default/local-pvc
$ ls
test-chain  test-chain-node0  test-chain-node1  test-chain-node2
```

7.3.2 添加普通节点

```
$ helm uninstall init-multi
release "init-multi" uninstalled
$ helm install increase-multi cita-cloud/cita-cloud-config --set config.action.
↪type=increaseMulti --set config.chainName=test-chain --set config.action.
↪increaseMulti.kmsPassword=123456 --set config.action.increaseMulti.node="192.168.10.
↪136:40000:node3" --set pvcName=local-pvc
NAME: increase-multi
LAST DEPLOYED: Thu Jul 29 23:38:26 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

生成文件

```
$ cd /tmp/hostpath-provisioner/default/local-pvc
$ ls
test-chain  test-chain-node0  test-chain-node1  test-chain-node2  test-chain-node3
```

多了节点文件夹 `test-chain-3`。

```
$ ls test-chain-node3
config.toml             executor-log4rs.yaml  kms-log4rs.yaml      node_config.toml
controller-log4rs.yaml  kms.db                network-log4rs.yaml  storage-log4rs.yaml
```

7.3.3 删除普通节点

```
$ helm uninstall increase-multi
release "increase-multi" uninstalled
$ helm install decrease-multi cita-cloud/cita-cloud-config --set config.action.
↪type=decreaseMulti --set config.chainName=test-chain --set config.action.
↪decreaseMulti.domain=node3
NAME: decrease-multi
LAST DEPLOYED: Thu Jul 29 23:42:38 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

文件变化

```
$ cd /tmp/hostpath-provisioner/default/local-pvc/
$ ls
test-chain  test-chain-node0  test-chain-node1  test-chain-node2
```

少了节点文件夹 `test-chain-node3`。

7.3.4 删除链

```
$ helm uninstall decrease-multi
release "decrease-multi" uninstalled
$ helm install clean cita-cloud/cita-cloud-config --set config.action.type=clean
Location: /home/mid/.kube/config
NAME: clean
LAST DEPLOYED: Thu Jul 29 23:47:11 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
$ helm uninstall clean
release "clean" uninstalled
```

文件变化

```
$ cd /tmp/hostpath-provisioner/default/local-pvc/
/tmp/hostpath-provisioner/default/local-pvc$ ls
/tmp/hostpath-provisioner/default/local-pvc$
```

节点有关文件夹全部被删除。

CHAPTER 8

隐私保护

CHAPTER 9

治理

本章节的操作需要超级管理员权限，且都非常危险，请谨慎操作。

超级管理员账户在运行 CITA-Cloud 时指定，详情参见快速入门相关章节的内容。

在 cloud-cli 中登录超级管理员账户：

```
$ cldi account login admin
OK, now the default user is `admin`, account addr is␣
↪0xae069e1925a1dad2a1f4c7034d87258dfd9b6532
```

9.1 更新超级管理员

超级管理员可以指定下一任超级管理员。

操作示例：

1. 查看系统配置，确认当前超级管理员账户。

```
$ cldi system-config
{
  "admin": "0xae069e1925a1dad2a1f4c7034d87258dfd9b6532",
  "block_interval": 6,
  "chain_id": "0x26b0b83e7281be3b117658b6f2636d0368cad3d74f22243428f5401a4b70897e",
  "validators": [
    "0xc35b3b7437a31b4d0a737041a17a8e181ae25ba5",
```

(下页继续)

(续上页)

```
    "0xa5e75c8ed90c17d2cd0b637943c7ce83248dbf20",
    "0x32872cec919211f5d144f8464b45140f4a146002",
    "0x790f590a1ea9764bcc26154c3de868ccf7bdcad4"
  ],
  "version": 0
}
```

1. 指定新的超级管理员

```
$ cldi update-admin 0x9817ac046e0b6a2903352d05497564147ddc0a6f
tx_hash: 0x1f3b25887bca912b5809e4860cd507574682efe457c01d2d450aa2067c06e826
```

1. 等待几秒，待交易上链之后，查看系统配置确认 `admin` 变成了新的账户地址。

```
$ cldi system-config
{
  "admin": "0x9817ac046e0b6a2903352d05497564147ddc0a6f",
  "block_interval": 6,
  "chain_id": "0x26b0b83e7281be3b117658b6f2636d0368cad3d74f22243428f5401a4b70897e",
  "validators": [
    "0xc35b3b7437a31b4d0a737041a17a8e181ae25ba5",
    "0xa5e75c8ed90c17d2cd0b637943c7ce83248dbf20",
    "0x32872cec919211f5d144f8464b45140f4a146002",
    "0x790f590a1ea9764bcc26154c3de868ccf7bdcad4"
  ],
  "version": 0
}
```

9.2 共识节点管理

设置新的共识节点列表。

操作示例：

1. 查看系统配置，确认当前的验证人列表。

```
$ cldi system-config
{
  "admin": "0xae069e1925a1dad2a1f4c7034d87258dfd9b6532",
  "block_interval": 3,
  "chain_id": "0x26b0b83e7281be3b117658b6f2636d0368cad3d74f22243428f5401a4b70897e",
  "validators": [
    "0x724f28ac8d069f150d541a07235ebc2363ae9b2a",
```

(下页继续)

(续上页)

```
    "0x1e0427b2a2ba34dceda5788d98f59aef0eb92f8e",
    "0x868a116bd018c3ed0facb4a761829485f03c2f73"
  ],
  "version": 0
}
```

1. 设置新的共识节点列表，新增地址为 0xc35b3b7437a31b4d0a737041a17a8e181ae25ba5 的共识节点。

```
$ cldi update-validators 0x724f28ac8d069f150d541a07235ebc2363ae9b2a␣
↪0x1e0427b2a2ba34dceda5788d98f59aef0eb92f8e␣
↪0x868a116bd018c3ed0facb4a761829485f03c2f73␣
↪0xc35b3b7437a31b4d0a737041a17a8e181ae25ba5
tx_hash: 0x54f9ebdb3d5e52b80cad92c7551a706a4ae20aecd7cde88c167f8659bec4a4b4
```

1. 等待几秒，待交易上链之后，查看系统配置确认 validators 变成了新的一组账户地址。

```
$ cldi system-config
{
  "admin": "0xae069e1925a1dad2a1f4c7034d87258dfd9b6532",
  "block_interval": 3,
  "chain_id": "0x26b0b83e7281be3b117658b6f2636d0368cad3d74f22243428f5401a4b70897e",
  "validators": [
    "0x724f28ac8d069f150d541a07235ebc2363ae9b2a",
    "0x1e0427b2a2ba34dceda5788d98f59aef0eb92f8e",
    "0x868a116bd018c3ed0facb4a761829485f03c2f73",
    "0xc35b3b7437a31b4d0a737041a17a8e181ae25ba5"
  ],
  "version": 0
}
```

共识节点列表变更之后，若有共识节点被剔除，则被剔除共识节点的共识会停止；若有新的共识节点被添加，则链可能会停止出块，直到新加入的共识节点启动并完成同步之后才会继续共识并出块。

9.3 修改出块间隔

超级管理员可以设置新的出块间隔。

操作示例：

1. 查看当前的出块间隔。

```
$ cldi system-config
{
  "admin": "0x9817ac046e0b6a2903352d05497564147ddc0a6f",
  "block_interval": 6,  //出块间隔为6秒
  "chain_id": "0x26b0b83e7281be3b117658b6f2636d0368cad3d74f22243428f5401a4b70897e",
  "validators": [
    "0xc35b3b7437a31b4d0a737041a17a8e181ae25ba5",
    "0xa5e75c8ed90c17d2cd0b637943c7ce83248dbf20",
    "0x32872cec919211f5d144f8464b45140f4a146002",
    "0x790f590a1ea9764bcc26154c3de868ccf7bdcad4"
  ],
  "version": 0
}
```

1. 修改出块间隔为 3 秒。

```
$ cldi  cldi set-block-interval 3
tx_hash: 0xa0cad608b1e0245f988d193380a56aea1056bca217f2a73dadb8dcec02e20cb9
```

1. 等待几秒，待交易上链之后，查看系统配置确认 `blockInterval` 变成了新的值。

```
$ cldi system-config
{
  "admin": "0x9817ac046e0b6a2903352d05497564147ddc0a6f",
  "block_interval": 3,
  "chain_id": "0x26b0b83e7281be3b117658b6f2636d0368cad3d74f22243428f5401a4b70897e",
  "validators": [
    "0xc35b3b7437a31b4d0a737041a17a8e181ae25ba5",
    "0xa5e75c8ed90c17d2cd0b637943c7ce83248dbf20",
    "0x32872cec919211f5d144f8464b45140f4a146002",
    "0x790f590a1ea9764bcc26154c3de868ccf7bdcad4"
  ],
  "version": 0
}
```

9.4 紧急制动

超级管理员在极端情况下的维护手段，开启紧急制动模式后，链上只接收超级管理员发送的治理交易，其他交易全部拒绝。（治理交易为本章的管理员操作）

主要使用场景为进行一些升级，维护等操作时，不希望有其他交易干扰。

9.4.1 开启

```
$  cldi emergency-brake on
tx_hash: 0x5b5aaa42b4312fb204a156d279e0f98805e62a98cdf9293c4d5f361fcefc3d88
```

等待几秒，待交易上链之后，将只有超级管理员可以发送治理交易，普通用户和普通交易会返回 `forbidden` 错误。

例如使用普通用户账户 `test` 创建合约：

```
$ cldi account login test
OK, now the default user is `test`, account addr is␣
↪0x415f568207900b6940477396fcd2c201efe49beb
$ cldi create␣
↪0x60806040523480156100105760008fd5b5060f58061001f6000396000f3006080604052600436106053576000357c010
thread 'main' panicked at 'called `Result::unwrap()` on an `Err` value: Status {␣
↪code: InvalidArgument, message: "Expect error: forbidden", metadata: MetadataMap {␣
↪headers: {"content-type": "application/grpc", "date": "Wed, 21 Jul 2021 09:05:37 GMT
↪"} }, source: None }', src/client.rs:146:14
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

9.4.2 关闭

```
$ cldi emergency-brake off
tx_hash: 0x347cf8a7ef89958e936344dc8318fd0ca7dd5eadaab60df74f094989549b7427
```

等待几秒，待交易上链之后，普通用户就可以再次正常发送交易了。

```
$ cldi account login test
OK, now the default user is `test`, account addr is␣
↪0x415f568207900b6940477396fcd2c201efe49beb
$ cldi create␣
↪0x60806040523480156100105760008fd5b5060f58061001f6000396000f3006080604052600436106053576000357c010
tx_hash: 0x7eeaefdf3d266a2b1a8b6350952b1353b230207bc4c089735a3d2e482dc1f77e
```

CHAPTER 10

数据管理

10.1 数据同步

10.2 数据迁移

10.3 数据裁剪

CHAPTER 11

运维

11.1 日志管理

日志在系统调试，问题定位，甚至业务运行方面都有着重要作用。CITA-Cloud 每个微服务的日志信息都会被记录到一个单独的日志文件。

11.1.1 日志位置

CITA-Cloud 的日志文件位于各个节点文件夹下的 `log` 文件中，每个微服务单独一个日志文件。

```
$ ls test-chain-0/logs
consensus-service.log   executor-service.log  network-service.log
controller-service.log  kms-service.log       storage-service.log
```

11.1.2 日志等级

日志等级定义如下：

- Warn: 警告信息
- Info: 默认信息等级
- Debug: 调试信息
- Trace: 最低等级

CITA-Cloud 会打印所设置等级及其以上的日志。

CITA-Cloud 默认的日志等级为 `info`.

```
$ cat controller-log4rs.yaml
# Set the default logging level and attach the default appender to the root
root:
  level: info
  appenders:
    - journey-service
```

`info` 等级打印出的日志

```
$ tail -10f /tmp/hostpath-provisioner/default/local-pvc/test-chain-0/logs/controller-
↪service.log
2021-08-03T06:17:06.966466511+00:00 INFO controller::chain - finalize_block: 3117,␣
↪block_hash: 0x78bc32f8c352cba5af069d4882048c8957b3011486eeaf152446667ad8a187c0
2021-08-03T06:17:06.967758688+00:00 INFO controller::node_manager - update node:␣
↪0x417181eb1961c70d19ce84cb7ffb102c8cfc2e2e
2021-08-03T06:17:06.967789307+00:00 INFO controller::node_manager - update node:␣
↪0xd8cd634b16a9aaa5ae9b4e3a181c80d70f0cbf56
2021-08-03T06:17:07.012242063+00:00 INFO controller::controller - add remote␣
↪proposal(0x8cc8d96407be8ae14c703c51dc8d8d19bd9e5c3321ede21b41c15937ea717e20)␣
↪through check_proposal
2021-08-03T06:17:09.949227372+00:00 INFO controller::controller - add remote␣
↪proposal(0xd8f56a1d049112a5a4a4cc8918a2beea9faa68d59ee2ad230733bf8a7c9e81f3)␣
↪through check_proposal
2021-08-03T06:17:09.969532670+00:00 INFO controller::node_manager - update node:␣
↪0x417181eb1961c70d19ce84cb7ffb102c8cfc2e2e
2021-08-03T06:17:09.971888525+00:00 INFO controller::util - height: 3118 hash␣
↪0xd8f56a1d049112a5a4a4cc8918a2beea9faa68d59ee2ad230733bf8a7c9e81f3
. . .
```

- 在生成节点配置时，可以通过`--is_stdout` 和`--log_level` 设置日志的输出和等级。详情参见配置说明章节的相关内容。

创建时设置打印日志的默认等级为 `Trace`，可以直接答应出 `Trace` 以及 `Trace` 级别以上的日志

```
$ helm install other-chain cita-cloud/cita-cloud-local-cluster --set pvcName=other-
↪pvc --set config.logLevel=Trace
$ tail -10f /tmp/hostpath-provisioner/default/other-pvc/other-chain-0/logs/controller-
↪service.log
2021-08-03T02:21:12.915282130+00:00 TRACE tracing::span::active - ->␣
↪Prioritize::queue_frame
2021-08-03T02:21:12.915284920+00:00 TRACE h2::proto::streams::prioritize - schedule_
↪send stream.id=StreamId(1303)
```

(下页继续)

(续上页)

```
2021-08-03T02:21:12.915287840+00:00 TRACE h2::proto::streams::store - Queue::push
2021-08-03T02:21:12.915290190+00:00 TRACE h2::proto::streams::store -  -> first entry
2021-08-03T02:21:12.915292720+00:00 TRACE tracing::span::active - <-␣
↪Prioritize::queue_frame
2021-08-03T02:21:12.915295570+00:00 TRACE tracing::span - -- Prioritize::queue_frame
2021-08-03T02:21:12.915298740+00:00 TRACE h2::proto::streams::prioritize - reserve_
↪capacity; stream.id=StreamId(1303) requested=1 effective=1 curr=0
. . .
```

- 运行过程中也可动态修改微服务的日志配置文件 `xxx-log4rs.yaml`。实现动态调整日志等级等。

11.2 异常排查

k8s 环境调试非常不便，且制作容器镜像时要考虑体积的问题，容器内缺乏调试用的工具和命令。

为了便于调试，我们在节点的 pod 中增加了一个专门用于调试的容器。

- 如果部署时使用cita_cloud_operator。可以通过设置`--need_debug` 参数为 `true` 开启该功能。默认用于调试的镜像是 `praqma/network-multitool`，包含的工具和命令详情参见链接。
- 如果部署时使用 Chart 工程，则不论是 `cita-cloud-local-cluster` 还是 `cita-cloud-multi-cluster-node`，都可以通过设置 `debug.enabled` 为 `true` 开启该功能，并且可以通过 `debug.imageName` 和 `debug.imageTag` 设置用于调试的容器镜像。

11.2.1 节点状况查询

- 使用 `kubectl` 的 `get` 命令直接查看当前节点的运行情况

```
$ kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
test-chain-0                7/7     Running   0          149m
test-chain-1                7/7     Running   0          149m
test-chain-2                7/7     Running   0          149m
```

- CITA-Cloud 使用 `kubectl` 的 `describe` 命令可以查看指定节点的详细状况。

```
$ kubectl describe pod test-chain-0
Name:         test-chain-0
Namespace:    default
Priority:     0
Node:         minikube/192.168.49.2
Start Time:   Mon, 02 Aug 2021 20:40:08 -0700
Labels:       app.kubernetes.io/instance=test-chain
```

(下页继续)

(续上页)

```
              app.kubernetes.io/managed-by=Helm
              app.kubernetes.io/name=cita-cloud-local-cluster
              app.kubernetes.io/version=6.1.0
              controller-revision-hash=test-chain-5b9f9b6b84
              helm.sh/chart=cita-cloud-local-cluster-6.1.0
              statefulset.kubernetes.io/pod-name=test-chain-0
Annotations:  <none>
Status:       Running
IP:           172.17.0.4
IPs:
  IP:           172.17.0.4
Controlled By:  StatefulSet/test-chain
Init Containers:
  cita-cloud-config:
    Container ID:  docker://
↪fc1b13ce38fe81c50c77843f866df71a20bffb9ad0ec58402ab22e42c668b7e0
    Image:         citacloud/cita_cloud_config:v6.0.0
    Image ID:      docker-pullable://citacloud/cita_cloud_
↪config@sha256:c148fd70cf28b886b0d23440625562bff559f16d4728be8bb256bc77855d2bfa
    Port:          <none>
    Host Port:     <none>
    Args:
      init
      --chain_name
      test-chain
      --super_admin
      0x415f568207900b6940477396fcd2c201efe49beb
      --is_bft
      true
      --work_dir
      /data
      --peers_count
      3
      --kms_password
      123456
      --is_stdout
      false
      --log_level
      info
    State:          Terminated
      Reason:       Completed
      Exit Code:    0
      Started:      Mon, 02 Aug 2021 20:40:12 -0700
```

(下页继续)

(续上页)

```
      Finished:     Mon, 02 Aug 2021 20:40:12 -0700
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /data from datadir (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-65x4l (ro)
Containers:
  debug:
    Container ID:   docker://
↪69a64a03102154f06625b3ef4bc26c2bb1d57498f01433d858422e88a07ac0c4
    Image:          praqma/network-multitool:latest
    Image ID:       docker-pullable://praqma/network-
↪multitool@sha256:7222852f7f120b44268f5bb8e2631bc667d740bd62dfc9eb695e89babd3e6d71
    Port:           9999/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Mon, 02 Aug 2021 20:40:14 -0700
    Ready:          True
    Restart Count:  0
    Environment:
      HTTP_PORT:  9999
      POD_NAME:   test-chain-0 (v1:metadata.name)
    Mounts:
      /data from datadir (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-65x4l (ro)
  network:
    Container ID:  docker://
↪b7b5eb98e9e2aabc7fcad6f3e07fb879ebd75015055ec628a287a138a1e7424c
    Image:         citacloud/network_direct:v6.0.0
    Image ID:      docker-pullable://citacloud/network_
↪direct@sha256:385580e69f9a68bea808b072614feecd50e079d7bf4be3aad616a738cab6062d
    Ports:         40000/TCP, 50000/TCP
    Host Ports:    0/TCP, 0/TCP
    Command:
      sh
      -c
      network run -p 50000 -k network_key
    State:          Running
      Started:      Mon, 02 Aug 2021 20:40:15 -0700
    Ready:          True
    Restart Count:  0
    Environment:
```

(下页继续)

(续上页)

```
      POD_NAME:  test-chain-0 (v1:metadata.name)
    Mounts:
      /data from datadir (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-65x4l (ro)
  consensus:
    Container ID:  docker://
↪d58537e6339069d33d9cf8e7b5216191705b5204fa556214d876dd3cf17830e4
    Image:         citacloud/consensus_bft:v6.0.0
    Image ID:      docker-pullable://citacloud/consensus_
↪bft@sha256:c56c1f873783bb6f2a4400d9718592c3121779fb4f682b8646482573e1c33853
    Port:          50001/TCP
    Host Port:     0/TCP
    Command:
      sh
      -c
      consensus run -p 50001
    State:          Running
      Started:      Mon, 02 Aug 2021 20:40:16 -0700
    Ready:          True
    Restart Count:  0
    Environment:
      POD_NAME:  test-chain-0 (v1:metadata.name)
    Mounts:
      /data from datadir (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-65x4l (ro)
  executor:
    Container ID:  docker://
↪e886f8b826061e5c5d5fe8980b2bb594f65933815432a159c87c015d96419a49
    Image:         citacloud/executor_evm:v6.1.0
    Image ID:      docker-pullable://citacloud/executor_
↪evm@sha256:25cde829576d6fa776e4c4022421da547152a628151ff038744fef2169da0892
    Port:          50002/TCP
    Host Port:     0/TCP
    Command:
      sh
      -c
      executor run -p 50002
    State:          Running
      Started:      Mon, 02 Aug 2021 20:40:17 -0700
    Ready:          True
    Restart Count:  0
    Environment:
      POD_NAME:  test-chain-0 (v1:metadata.name)
```

(下页继续)

(续上页)

```
    Mounts:
      /data from datadir (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-65x4l (ro)
  storage:
    Container ID:  docker://
↪1ec6ecf500b3f046511dd3c82f9db887f99028f7d5945924e9e4b75fcb35c105
    Image:         citacloud/storage_rocksdb:v6.1.0
    Image ID:      docker-pullable://citacloud/storage_
↪rocksdb@sha256:747f445b36e0e747212742de6ebc37e59f19a21fc2b6f5c6d7f1b8e8f8aba433
    Port:          50003/TCP
    Host Port:     0/TCP
    Command:
      sh
      -c
      storage run -p 50003
    State:          Running
      Started:      Mon, 02 Aug 2021 20:40:19 -0700
    Ready:          True
    Restart Count:  0
    Environment:
      POD_NAME:  test-chain-0 (v1:metadata.name)
    Mounts:
      /data from datadir (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-65x4l (ro)
  controller:
    Container ID:  docker://
↪2a661310c5d392496bf2721c26e98af8412c790276e0258171025ff0e4154ada
    Image:         citacloud/controller:v6.1.0
    Image ID:      docker-pullable://citacloud/
↪controller@sha256:b0908004f6ea1a20c22f2d51a9f2e7e179348f5ebd68339c6c596934a8b8f761
    Port:          50004/TCP
    Host Port:     0/TCP
    Command:
      sh
      -c
      controller run -p 50004
    State:          Running
      Started:      Mon, 02 Aug 2021 20:40:20 -0700
    Ready:          True
    Restart Count:  0
    Environment:
      POD_NAME:  test-chain-0 (v1:metadata.name)
    Mounts:
```

(下页继续)

(续上页)

```
      /data from datadir (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-65x4l (ro)
  kms:
    Container ID:  docker://
↪57a48e2f7fd968617c194499c31113c8746ba78260b5d9bdec0ab35336c20d6b
    Image:         citacloud/kms_sm:v6.0.0
    Image ID:      docker-pullable://citacloud/kms_
↪sm@sha256:ae41f2d99d26f6b684f04f4c0e4c956db6c8984673a29e752673c4aed40f64da
    Port:          50005/TCP
    Host Port:     0/TCP
    Command:
      sh
      -c
      kms run -p 50005 -k /kms/key_file
    State:          Running
      Started:      Mon, 02 Aug 2021 20:40:22 -0700
    Ready:          True
    Restart Count:  0
    Environment:
      POD_NAME:  test-chain-0 (v1:metadata.name)
    Mounts:
      /data from datadir (rw)
      /kms from kms-key (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-65x4l (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  kms-key:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  test-chain-kms-secret
    Optional:    false
  datadir:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the␣
↪same namespace)
    ClaimName:  test-pvc
    ReadOnly:   false
  default-token-65x4l:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-65x4l
```

(下页继续)

(续上页)

```
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  2m18s  default-scheduler  Successfully assigned default/test-
↪chain-0 to minikube
  Normal  Pulled     2m15s  kubelet            Container image "citacloud/cita_cloud_
↪config:v6.0.0" already present on machine
  Normal  Created    2m14s  kubelet            Created container cita-cloud-config
  Normal  Started    2m13s  kubelet            Started container cita-cloud-config
  Normal  Pulled     2m13s  kubelet            Container image "praqma/network-
↪multitool:latest" already present on machine
  Normal  Started    2m12s  kubelet            Started container debug
  Normal  Created    2m12s  kubelet            Created container debug
  Normal  Pulled     2m12s  kubelet            Container image "citacloud/network_
↪direct:v6.0.0" already present on machine
  Normal  Created    2m12s  kubelet            Created container network
  Normal  Started    2m11s  kubelet            Started container network
  Normal  Pulled     2m11s  kubelet            Container image "citacloud/consensus_
↪bft:v6.0.0" already present on machine
  Normal  Created    2m11s  kubelet            Created container consensus
  Normal  Pulled     2m10s  kubelet            Container image "citacloud/executor_
↪evm:v6.1.0" already present on machine
  Normal  Started    2m10s  kubelet            Started container consensus
  Normal  Created    2m9s   kubelet            Created container executor
  Normal  Started    2m9s   kubelet            Started container executor
  Normal  Pulled     2m9s   kubelet            Container image "citacloud/storage_
↪rocksdb:v6.1.0" already present on machine
  Normal  Created    2m8s   kubelet            Created container storage
  Normal  Started    2m7s   kubelet            Started container storage
  Normal  Pulled     2m7s   kubelet            Container image "citacloud/
↪controller:v6.1.0" already present on machine
  Normal  Created    2m6s   kubelet            Created container controller
  Normal  Started    2m5s   kubelet            Started container controller
  Normal  Pulled     2m5s   kubelet            Container image "citacloud/kms_sm:v6.0.
↪0" already present on machine
  Normal  Created    2m4s   kubelet            Created container kms
  Normal  Started    2m4s   kubelet            Started container kms
```

11.3 服务监控

CITA-Cloud 有针对节点的监控组件CITA-Monitor。

开启方法：

- 如果部署时使用cita_cloud_operator。可以通过设置--need_monitor 参数为 true 开启该功能。

CHAPTER 12

架构设计

12.1 整体设计原则

CITA-Cloud 总体设计上依然采用微服务架构，划分为 Controller，Network，Consensus，Storage，Executor，KMS 六个微服务。微服务之间相互解耦，达到不同实现可以灵活替换，自由组合的目的。

解耦设计的细节参见底层链技术白皮书。

在此基础上，为了能够快速构建起完整的，成熟的生态。在之前解耦的基础上，让解耦出的每一个微服务都能独立完成某项功能，每个微服务的接口能够自洽，方便直接复用已有的库或者软件。

12.2 协议详解

12.2.1 Network

Network 微服务，主要提供网络部分的功能，分为收/发两大部分功能。收的部分采用了控制反转，收到的网络报文根据报文头中的字段分发到不同的 Grpc 地址，因此只提供了一个注册接口 (RegisterNetworkMsgHandler); 发的部分提供了单播 (SendMsg) 和广播 (Broadcast) 两个接口。此外就是一个查询网络连接状态的接口 (GetNetworkStatus)。

实现上就是已有网络库的简单封装。

其中一个实现network_direct直接使用系统网络库，内部实现为节点的全互联。另外一个实现network_p2p，使用了已有的一个 p2p 库。

12.2.2 Storage

Storage 微服务，主要提供 KV 存储相关的功能，涵盖了常用的增删改查功能。接口上有：Store(其语义是 updata，同时包含增和改的功能)，Load，Delete。

针对区块链业务，预先定义了不同的 region：

```
enum Regions {
    GLOBAL = 0;
    TRANSACTIONS = 1;
    HEADERS = 2;
    BODIES = 3;
    BLOCK_HASH = 4;
    PROOF = 5;
    RESULT = 6;
    TRANSACTION_HASH2BLOCK_HEIGHT = 7;
    BLOCK_HASH2BLOCK_HEIGHT = 8;  // In SQL db, reuse 4
    TRANSACTION_INDEX = 9;
    BUTTON = 10;
}
```

用户可以根据自己的需要调整 region 列表。

具体实现上，就是已有存储系统的简单封装。

当前实现有基于 sqlite 的storage_sqlite，基于 rocksdb 的storage_rocksdb，以及基于 tikv 的storage_tikv。

region 的实现，基于 sqlite 时是用不同的 table 来实现；基于 tikv 时是直接把 region 序列化之后拼接在 key 的前面；基于 rocksdb 时是用不同的 ColumnFamily 来实现。

12.2.3 KMS

KMS 微服务，主要提供私钥加密存储，以及相关的密码学服务。接口有：GenerateKeyPair，HashData，VerifyDataHash，SignMessage，RecoverSignature，形成——生成密钥对/哈希以及相关的验证/签名以及相关的验证——这么一个自洽的功能集合。此外还有一个查询接口 GetCryptoInfo。可以认为是一个软件加密机加上一个密码学工具箱。

目前的实现kms_eth和kms_sm，分别实现了 secp256k1+keccak 和 sm2+sm3 两种密码学算法组合。

12.2.4 Executor

Executor 微服务，主要提供根据交易改变链上状态以及查询链上状态的功能。接口有：Exec 和 Call，分别对应执行和查询。

至于执行的细节，比如采用何种 VM；状态有哪些内容；状态如何组织和保存，都由具体实现来决定。

后续会给出一些兼容已有链的特定实现，比如executor_chaincode就是兼容 Fabric 的实现。executor_evm则是兼容以太坊的实现。

也可以针对一些特定应用场景，提供特定的 VM 和智能合约编程语言。比如可信计算，隐私计算，数据格式转换等。

甚至可以不提供智能合约，直接针对具体应用实现，类似于原生合约。

12.2.5 Consensus

Consensus 微服务，主要提供让提案在多个共识参与方之间达成一致的功能。接口包括：

1. 实现在 controller 中的：GetProposal 本节点提交的提案；CheckProposal 检查别的节点提交的提案；CommitBlock 确认一个提案。
2. 实现在 Consensus 中的：CheckBlock 检查同步自别的节点的已经确认的提案；Reconfigure 处理系统配置变更。

单独这个微服务的功能，可以认为是一个分歧解决机。

实现上，目前尝试了 PoX 类型的consensus_proof_of_sleep，基于 raft 的consensus_raft，以及基于 bft 的consensus_bft。

12.2.6 Controller

Controller 微服务在整个区块链中处于核心的位置，主导所有主要的流程，并给上层用户提供 RPC 接口。

接口除了前述的针对 Consensus 微服务的接口，就是针对上层用户的 RPC 接口。其中最重要的是 SendRawTransaction 发送交易接口，剩下的都是一些信息查询接口。

单独就这个微服务来说，可以认为是一个提案管理系统。用户通过发送交易接口，提交原始交易数据，Controller 管理这些原始交易数据。通过计算原始交易数据的哈希，组装 CompactBlock，以及再次哈希，形成 Consensus 需要的提案，管理这些提案。这里所说的管理，包括持久化，同步，以及验证其合法性。

Blockchain.proto 文件中定义了一套交易和块的数据结构，但是前面所述的从原始交易数据如何产生最终 Consensus 需要的提案，并且这个过程还是要可验证的，这些都由具体实现决定。未来我们会提供一个框架，方便用户自定义整个流程，甚至是自定义交易和块等核心数据结构。

实现上，目前只有一个controller_poc。主要模块还是以自己实现为主，只有同步模块使用了 `Syncthing`。实现上的技术细节参见项目 Wiki。整个实现的代码量还是比较大，后续会考虑进一步细化设计，让尽量多的模块能够复用已有的软件或者库。

CHAPTER 13

RPC 列表

13.1 GetPeerCount

当前节点连接数。

- 参数:

  无。

- 返回值

  `uint64` - 本节点连接节点个数。

- 示例

```
$ cldi peer-count
peer_count: 2
```

13.2 GetBlockNumber

返回当前块高度。

- 参数

  `bool` - `true` 表示获取 `Pending` 状态的块高；`false` 表示获取 `latest` 状态的块高。

- 返回值

uint64 - 当前块高度。

- 示例

```
$ cldi block-number -p
block_number: 58014
```

cloud-cli 的-p 参数表示获取 Pending 状态的块高；不加该参数表示获取 latest 状态的块高。

13.3 GetTransaction

根据交易哈希查询交易。

- 参数

  bytes - 交易哈希值。

- 返回值

  RawTransaction - 交易结构体。

- 示例

```
$ cldi get-tx 0x22ce8fe4e68e791825edad6cd7a944e77eba3e8d41fe582bcbf7d3b06fb17623
tx: {
  "transaction": {
    "transaction": {
      "chain_id": "0x22ae4cd3acabd1f259e255dba07a2e463cea57cf7d2802ebc399330f0bb18b02
↪",
      "data": "0x4f2be91f",
      "nonce": "727992510686994504",
      "quota": 3000000,
      "to": "0x253479ef7f0209ad761960e9f41bb18d1113b2bb",
      "valid_until_block": 22629,
      "value": "0x0000000000000000000000000000000000000000000000000000000000000000",
      "version": 0
    },
    "transaction_hash":
↪"0x22ce8fe4e68e791825edad6cd7a944e77eba3e8d41fe582bcbf7d3b06fb17623",
    "witness": {
      "sender": "0x415f568207900b6940477396fcd2c201efe49beb",
      "signature":
↪"0x947ce32efb441bb9a8240f256700df69fb8ea71af94f5a93bf31a0171fbd72b07999d30d0b076c2ecaa19d0327673cae
↪"
    }
  },
```

(下页继续)

(续上页)

```
  "type": "Normal"
}
```

13.4 GetSystemConfig

查询链上系统配置数据。

- 参数

  无。

- 返回值

  `SystemConfig` - 系统配置结构体。

- 示例

```
$ cldi system-config
{
  "admin": "0x642741f75dad495e75c20ab6fd4e84b3f5469b23",
  "block_interval": 3,
  "chain_id": "0x22ae4cd3acabd1f259e255dba07a2e463cea57cf7d2802ebc399330f0bb18b02",
  "validators": [
    "0x3fbef0cc8aac891279520d148188ebdb156bf70e",
    "0x0e952599fcb4c9235ddd8ba36d96bfdd878d79bf",
    "0x106013370a48bc1988a1d5733d5ecf9a85d3f721"
  ],
  "version": 0
}
```

13.5 GetVersion

获取当前软件的版本号。

- 参数

  无。

- 返回值

  `String` - 软件版本号。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto/protos/controller.proto \
    -import-path ~/cita_cloud_proto/protos \
    `minikube ip`:30004 controller.RPCService/GetVersion
{
  "version": "6.1.0"
}
```

13.6 GetBlockHash

获取指定块高的块哈希值。

- 参数

  `uint64` - 块高度。

- 返回值

  `bytes` - 块哈希值。

- 示例

```
$ cldi block-hash 22629
hash: 0x5a8747d1b1f4c8ba53ca7b01b33f4e2044974a26b4a239bf3ab8df6dadef0f89
```

13.7 GetTransactionBlockNumber

获取指定交易所在的块高度。

- 参数

  `bytes` - 交易哈希值。

- 返回值

  `uint64` - 块高度。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '{"hash": "fJkDwCaMi7mOnHTVQ/
↪IcGNr83aoUxnj5kAkDDhRkya0="}' \
   -proto ~/cita_cloud_proto-master/protos/controller.proto \
   -import-path ~/cita_cloud_proto-master/protos \
   `minikube ip`:30004 controller.RPCService/GetTransactionBlockNumber
{
```

(下页继续)

(续上页)

```
  "blockNumber": "22"
}
```

13.8 GetTransactionIndex

获取指定交易在块中的序号。

- 参数

  bytes - 交易哈希值。

- 返回值

  uint64 - 序号。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '{"hash": "fJkDwCaMi7mOnHTVQ/
↪IcGNr83aoUxnj5kAkDDhRkya0="}' \
    -proto ~/cita_cloud_proto-master/protos/controller.proto \
    -import-path ~/cita_cloud_proto-master/protos \
    `minikube ip`:30004 controller.RPCService/GetTransactionIndex
{
  "txIndex": "2"
}
```

13.9 GetTransactionCount

获取账户发出的交易数量

- 参数

  bytes-用户地址

- 返回值

  nonce-用户发出交易的数量

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '{"address": "AQEBAQEBAQEBAQEBAQEBAQEBAQEB"}
↪' \
    -proto ~/cita_cloud_proto/protos/vm/evm.proto \
    -import-path ~/cita_cloud_proto/protos \
    localhost:50002 evm.RPCService/GetTransactionCount
```

(下页继续)

(续上页)

```
{
  "nonce": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="
}
```

13.10 GetBlockByHash

根据块哈希值查询块。

- 参数

  bytes - 块哈希值。

- 返回值

  CompactBlock - 块结构体。

- 示例

```
$ cldi get-block -h 0x5a8747d1b1f4c8ba53ca7b01b33f4e2044974a26b4a239bf3ab8df6dadef0f89
{
  "height": 22629,
  "prev_hash": "0xc44215f6537d36330b30815fcf9603824ed861ac1f13ba38670239bd3fbaff9b",
  "proposer": "0x0e952599fcb4c9235ddd8ba36d96bfdd878d79bf",
  "timestamp": "2021-07-27 20:26:53.297 -07:00",
  "transaction_root":
↪"0x1ab21d8355cfa17f8e61194831e81a8f22bec8c728fefb747ed035eb5082aa2b",
  "tx_count": 0,
  "tx_hashes": [],
  "version": 0
}
```

13.11 GetBlockByNumber

根据块高度查询块。

- 参数

  uint64 - 块高度。

- 返回值

  CompactBlock - 块结构体。

- 示例

```
$ cldi get-block -n 22629
{
  "height": 22629,
  "prev_hash": "0xc44215f6537d36330b30815fcf9603824ed861ac1f13ba38670239bd3fbaff9b",
  "proposer": "0x0e952599fcb4c9235ddd8ba36d96bfdd878d79bf",
  "timestamp": "2021-07-27 20:26:53.297 -07:00",
  "transaction_root":
↪"0x1ab21d8355cfa17f8e61194831e81a8f22bec8c728fefb747ed035eb5082aa2b",
  "tx_count": 0,
  "tx_hashes": [],
  "version": 0
}
```

13.12 SendRawTransaction

发送交易。

- 参数

  `RawTransaction` - 交易结构体。

- 返回值

  `bytes` - 交易哈希值。

- 示例

  `cli` 工具会根据用户传递的参数组装 RawTransaction 结构。

```
$ cldi send -t 0x253479ef7f0209ad761960e9f41bb18d1113b2bb 0x4f2be91f
tx_hash: 0x9684414367bd24f06dc129c097186e4a1668c0888354d32ce58e9aeb199397e3
```

13.13 GetTransactionReceipt

获取交易回执。

- 参数

  `bytes` 交易哈希值。

- 返回值

  `Receipt`-交易回执。

- 示例

```
$ cldi receipt 0xf62b9116c0a62a592a2fee55e714e1501fb84075770c2fbff0b484c75f61e405
{
  "block_number": 22419,
  "contract_addr": "0x253479ef7f0209ad761960e9f41bb18d1113b2bb",
  "cumulative_quota_used":
↪"0x0000000000000000000000000000000000000000000000000000000000018ed3",
  "error_msg": "",
  "legacy_cita_block_hash":
↪"0xdc7457b2c12dc0105aa7d0355d889911e6e10c2124958219b46b72ec6e9a5e28",
  "logs": [],
  "logs_bloom":
↪"0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
  "quota_used": "0x0000000000000000000000000000000000000000000000000000000000018ed3",
  "state_root": "0xf777eab35bc620de67c0ecab6c460f12047f7f7f828b1fb336959045ffa1972c",
  "tx_hash": "0xf62b9116c0a62a592a2fee55e714e1501fb84075770c2fbff0b484c75f61e405",
  "tx_index": 0
}
```

13.14 GetCode

- 参数

  bytes-合约地址

- 返回值

  bytes-二进制代码

- 示例

```
$ cldi get-code 0x253479ef7f0209ad761960e9f41bb18d1113b2bb
code:␣
↪0x6080604052600436106053576000357c0100000000000000000000000000000000000000000000000000000000900463
```

13.15 GetBalance

获取账户余额。

- 参数

  bytes-账户地址

- 返回值

Balance-账户余额

- 示例

```
$ cldi get-balance 0x415f568207900b6940477396fcd2c201efe49beb
balance: 0x0000000000000000000000000000000000000000000000000000000000000000
```

13.16 GetAbi

获取合约的 Abi

- 参数

  bytes-合约地址

- 返回值

  ByteAbi-Abi

- 示例

```
$ cldi get-abi 0x253479ef7f0209ad761960e9f41bb18d1113b2bb
ABI:
[
        {
                "constant": true,
                "inputs": [],
                "name": "count",
                "outputs": [
                        {
                                "name": "",
                                "type": "uint256"
                        }
                ],
                "payable": false,
                "stateMutability": "view",
                "type": "function"
        },
        {
                "constant": false,
                "inputs": [],
                "name": "add",
                "outputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "function"
```

(下页继续)

(续上页)

```
        },
        {
                "constant": false,
                "inputs": [],
                "name": "reset",
                "outputs": [],
                "payable": false,
                "stateMutability": "nonpayable",
                "type": "function"
        }
]
```

CHAPTER 14

版本

14.1 v3.0.0

版本发布说明

14.2 v4.0.0

版本发布说明

14.3 v5.0.0

版本发布说明

14.4 v6.0.0

版本发布说明

14.5 v6.1.0

版本发布说明

CHAPTER 15

中间件

15.1 分布式身份

参见分布式数字身份产业联盟。

15.2 数据存证

参见RivLink。

15.3 物联网设备

参见RivIoT。

15.4 数据协作

参见RivFlow。

15.5 跨链连接

15.6 隐私计算

CHAPTER 16

常见问题

CHAPTER 17

词汇表

17.1 区块链

- 区块

一个区块是一个数据包，其中包含零个或多个交易，父块的哈希值，以及其他数据。除了初始的“创世区块”以外每个区块都包含它父块的哈希值，区块的全部集合被称为区块链，并且包含了一个网络里的全部交易历史。

- 账户

账户是在总账中的记录，由它的地址来索引，总账包含有关该账户的状态的完整的数据。在一个货币系统里，这包含了货币余额。

- 挖矿

通过暴力尝试来找到一个字符串，使得它加上一组交易信息后的哈希值符合特定规则（例如前缀包括若干个0），找到的人可以宣称新区块被发现，并获得系统奖励。

- 矿工

参与挖矿的人或组织。

- 矿机

专门为挖矿而设计的设备，包括基于软件、GPU、FPGA 和专用芯片等多种实现。

- 矿池

采用团队协作方式来集中算力进行挖矿，对产出的虚拟货币进行分配。

- P2P

点到点的通信网络，网络中所有节点地位均等，不存在中心化的控制机制。

- 去中心化

不存在第三方中心机构，全体网民可以共同参与、权级相同。

- 分布式账本

可以在多个站点、不同地理位置或者多个机构组成的网络里进行分享的资产数据库。

- EVM

以太坊的虚拟机，智能合约运行的环境。

- FLP Impossibility

FLP 不可能性。分布式领域的一个著名理论，它的结论是：在网络可靠，存在节点失效（即便只有一个）的最小化异步模型系统中，不存在一个可以解决一致性问题的确定性算法。

- CPA

分布式计算系统不可能同时确保一致性（Consistency）、可用性（Availability）和分区容忍性（Partition），设计中往往需要弱化对某个特性的保证。一致性指等同于所有节点访问同一份最新的数据副本；可用性指每次请求都能获取到非错的响应，但是不保证获取的数据为最新数据；分区容忍性指以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在 C 和 A 之间做出选择。

- 女巫攻击

少数节点通过伪造或盗用身份伪装成大量节点，进而对分布式系统进行破坏。

- UTXO

未花费的输出（Unspent Transaction Outputs）。每一笔交易的输入都会引用先前有效交易的未花费输出，再创建新的未花费输出。账户余额就是发往该账户地址的一笔笔未花费输出的累加。

17.2 密码学

密码学是数学和计算机科学的分支，研究如何在有敌人存在的环境中安全通信。

- 哈希

即散列算法，将任意长度的二进制值映射为较短的固定长度的二进制值的算法。

- 非对称加密

一种特殊的加密，具有在同一时间生成两个密钥的处理（通常称为私钥和公钥），使得利用一个密钥对文档进行加密后，可以用另一个密钥进行解密。一般地，公钥是公开的，私钥自己保留。

- 数字签名

用户用私钥为文档产生一段叫做签名的短字符串数据，任何拥有相应公钥的人都可以验证该签名。

- CA

Certificate Authority，负责证书的创建、颁发，在 PKI 体系中最为核心的角色。

- PKI

Public Key Infrastructure，基于公钥体系的安全基础设施，是一组由硬件、软件、参与者、管理政策与流程组成的基础架构，其目的在于创造、管理、分配、使用、存储以及撤销数字证书。

- Zero-knowledge Proofs

零知识证明，一种密码学技术，允许两方（证明者和验证者）来证明某个提议是真实的，而且无需泄露除了它是真实的之外的任何信息。

- Merkle Tree

中文名叫默克尔树，又叫哈希树，是一种二叉树，由一个根节点、一组中间节点和一组叶节点组成。最下面的叶节点包含存储数据或其哈希值，每个中间节点是它的两个孩子节点内容的哈希值，根节点也是由它的两个子节点内容的哈希值组成。

17.3 CITA-Cloud

CITA-Cloud 是一个基于云原生的联盟链框架，用户可以基于该框架快速定制出一个适合具体业务场景的链。

- 智能合约

一段代码，被部署在分享的、复制的账本上，它可以维持自己的状态，控制自己的资产和对接收到的外界信息或者资产进行回应。

- 微服务

即 CITA-Cloud 将区块链节点的各功能解耦之后的组件，如共识，存储，点对点网络协议等这些都是微服务。微服务架构能够有效提高节点处理能力，并且使 CITA-Cloud 非常容易水平扩展。

- log4rs

log4rs 是一个高度可配置的日志记录框架，模仿 Java 的 Logback 和 log4j 库。

17.4 共识

- PoW

Proof of Work，工作量证明。

- PoS

Proof of Stake，权益证明。

- PBFT

Practical Byzantine Fault Tolerance，实用拜占庭容错算法。

- Raft 算法

一种非拜占庭容错的共识算法，比 Paxos 更容易理解。

- 拜占庭将军问题

LESLIE LAMPORT 为了更形象的介绍分布式系统共识问题，而杜撰的一个故事。拜占庭帝国军队的将军们必须全体一致的决定是否攻击某一支敌军。问题是这些将军在地理上是分隔开来的，并且将军中存在叛徒。叛徒可以任意行动以达到以下目标：欺骗某些将军采取进攻行动；促成一个不是所有将军都同意的决定，如当将军们不希望进攻时促成进攻行动；或者迷惑某些将军，使他们无法做出决定。如果叛徒达到了这些目的之一，则任何攻击行动的结果都是注定要失败的，只有完全达成一致的努力才能获得胜利。

17.5 Kubernetes

Kubernetes 是一个可移植的、可扩展的开源平台，用于管理容器化的工作负载和服务，可促进声明式配置和自动化。Kubernetes 拥有一个庞大且快速增长的生态系统。Kubernetes 的服务、支持和工具广泛可用。

- kubectl

Kubectl 命令行工具可以用来管理 Kubernetes 集群。

- minikube

minikube 是一种可以在本地运行 kubernetes 的工具，帮助我们更加轻松的搭建和使用 kubernetes。

- PV

持久卷（PersistentVolume，PV）是集群中的一块存储，可以由管理员事先供应，或者使用存储类（Storage Class）来动态供应。持久卷是集群资源，就像节点也是集群资源一样。PV 持久卷和普通的 Volume 一样，也是使用卷插件来实现的，只是它们拥有独立于任何使用 PV 的 Pod 的生命周期。此 API 对象中记述了存储的实现细节，无论其背后是 NFS、iSCSI 还是特定于云平台的存储系统。

- PVC

持久卷申领（PersistentVolumeClaim，PVC）表达的是用户对存储的请求。概念上与 Pod 类似。Pod 会耗用节点资源，而 PVC 申领会耗用 PV 资源。Pod 可以请求特定数量的资源（CPU 和内存）；同样 PVC 申领也可以请求特定的大小和访问模式（例如，可以要求 PV 卷能够以 ReadWriteOnce、ReadOnlyMany 或 ReadWriteMany 模式之一来挂载，参见访问模式）。

17.6 Docker

Docker 是开发、运输和运行应用程序的开放平台。Docker 使我们能够将应用程序与基础结构分开，以便快速交付软件。 有了 Docker，我们可以以与管理应用程序相同的方式管理基础结构。通过利用 Docker 的快速运输、测试和部署代码的方法，我们可以显著减少编写代码和在生产中运行代码之间的延迟。

17.7 云原生

云原生以微服务，容器，容器编排为主要技术，目的是构建一个完善的基础设施，可以方便的在云这种新型的动态环境中构建和部署应用。

2015 年由 Google 、Redhat 以及微软等大型云计算厂商以及一些开源公司共同牵头成立了 CNCF ——云原生基金会。谷歌和 Redhat 发布的 Kubernetes 成为了 CNCF 托管的第一个开源项目。