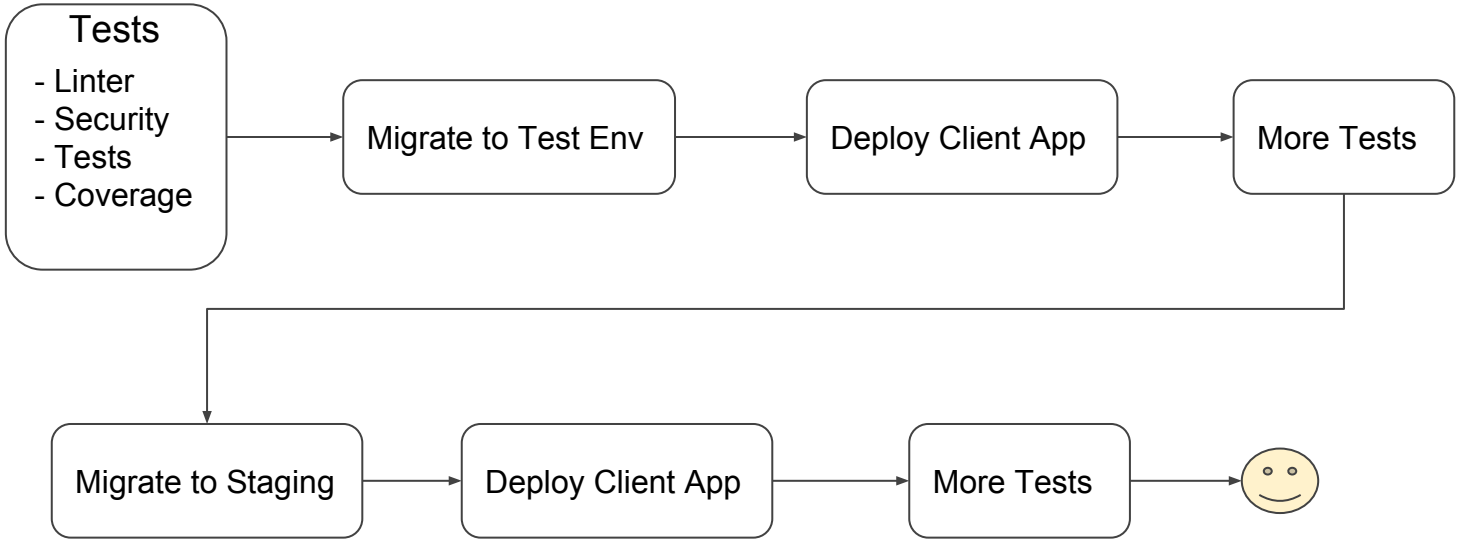
CI for Smart Contract Development on Ethereum

Miguel Hervas Lazaro
Software Engineer
@ Centrifuge

Problem Statement

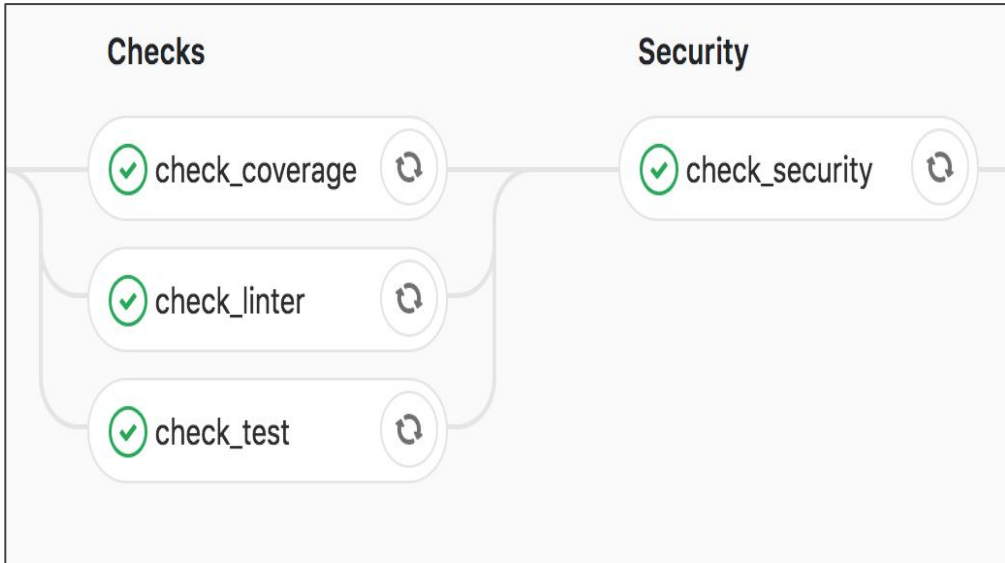
- Difficult to ensure reliable releases of new smart contract versions
- Many manual steps
  - Starting up ethereum client nodes
  - Unlocking accounts
  - Fetching account keys
- Maintain multiple Smart Contract versions
- Publishing ABIs and Addresses

Overview Diagram Flow



Smart Contract Checks

- Truffle Tests
- Test Coverage - [**solidity-coverage** + **coveralls**]
- Linters - [**solhint**]
- Security Analysis - [**Mythril OSS**]
- All run in Gitlab CI



Smart Contract Checks - gitlab-ci.yml

```yaml
check_test:
  stage: checks
  # This way, we can start additional services required by our
  # Here, we start a latest Parity node with a dev chain, to b
  services:
  - name: parity/parity
    alias: parity
    command: ["--chain=dev", "--jsonrpc-interface=0.0.0.0"]
  # This is how we expose environment variables to our build s
  # You would also need to have `ETH_ACCOUNT` set up in order
  # We recomment setting that variable as a build variable in
  #   https://gitlab.com/<your/project>/settings/ci_cd
  # Please see https://docs.gitlab.com/ee/ci/variables/ for mo
  variables:
    PROVIDER_ENDPOINT: "http://parity:8545"
    # This is base64-wrapped account info, to be used in Truff
    # For actual testnet deployment you'd need to setup `ETH_A
    ETH_ACCOUNT: "eyJpZCI6IjliYTdmNDVlLTlhYTItN2IwZS1iMDI2LWQw
  script:
    - truffle migrate --network dev
    - truffle test --network dev
    - npm test
  # Artifact are a way to pass a build results to a subsequent
  # They are not unlike build cache, but need to be explicitly
  # See the `dependencies:` section in the `build_sources`
  artifacts:
    paths:
      - build/
```

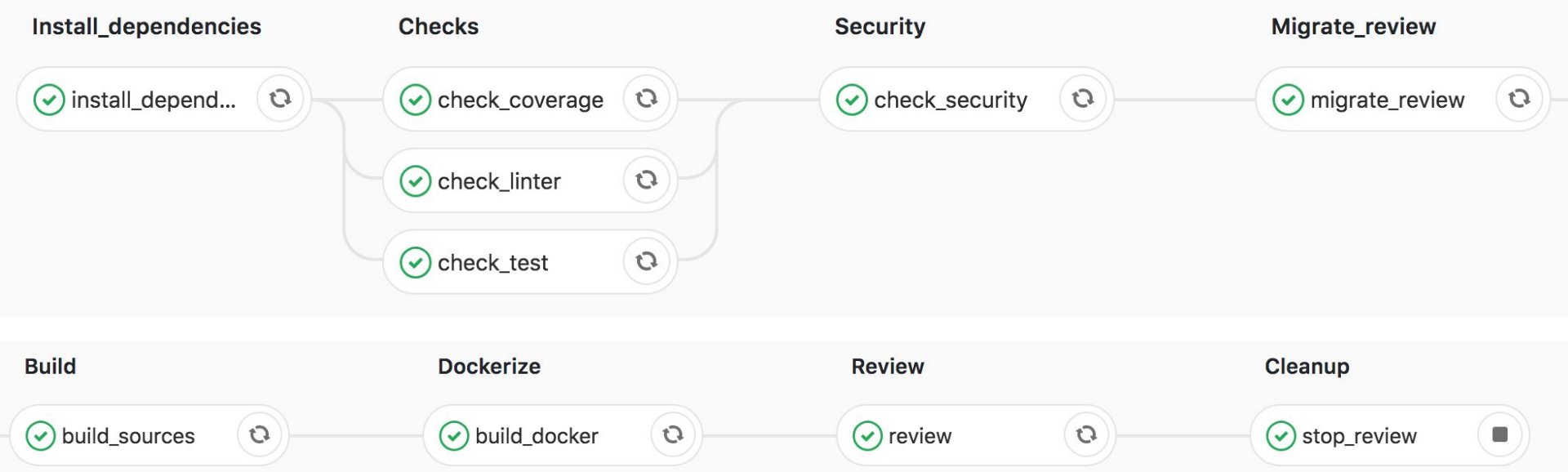
```yaml
check_linter:
  stage: checks
  cache: {}
  before_script:
    - npm install -g solhint
  script:
    - echo "Running solidity linter"
    - solhint "contracts/**/*.sol"
  allow_failure: true

check_security:
  stage: security
  image: docker:stable
  cache: {}
  variables:
    DOCKER_DRIVER: overlay
  services:
    - docker:dind
  script:
    - docker run -v $(pwd):/tmp -w "/tmp/" mythril/myth --truffle
  dependencies:
    - check_test
```

CI Pipeline definition

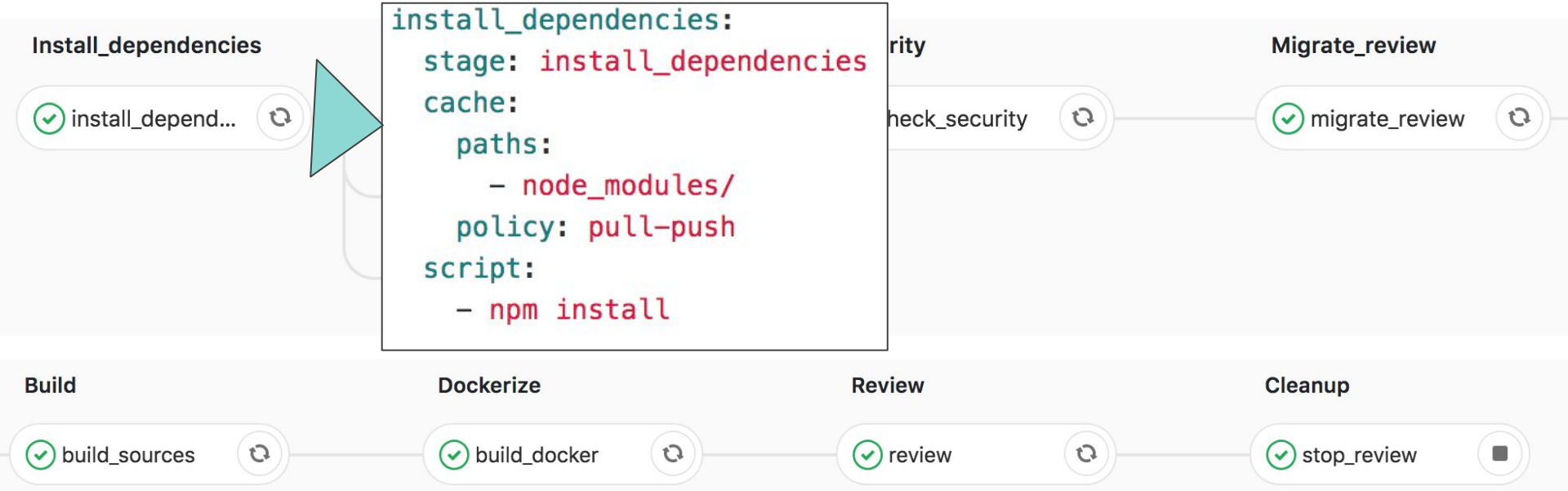
Assumptions:

- Developers work on feature branch
- Pull Request against master branch
- Standalone client app
- On demand review environments before merging to master
- Master branch represents staging+ environments

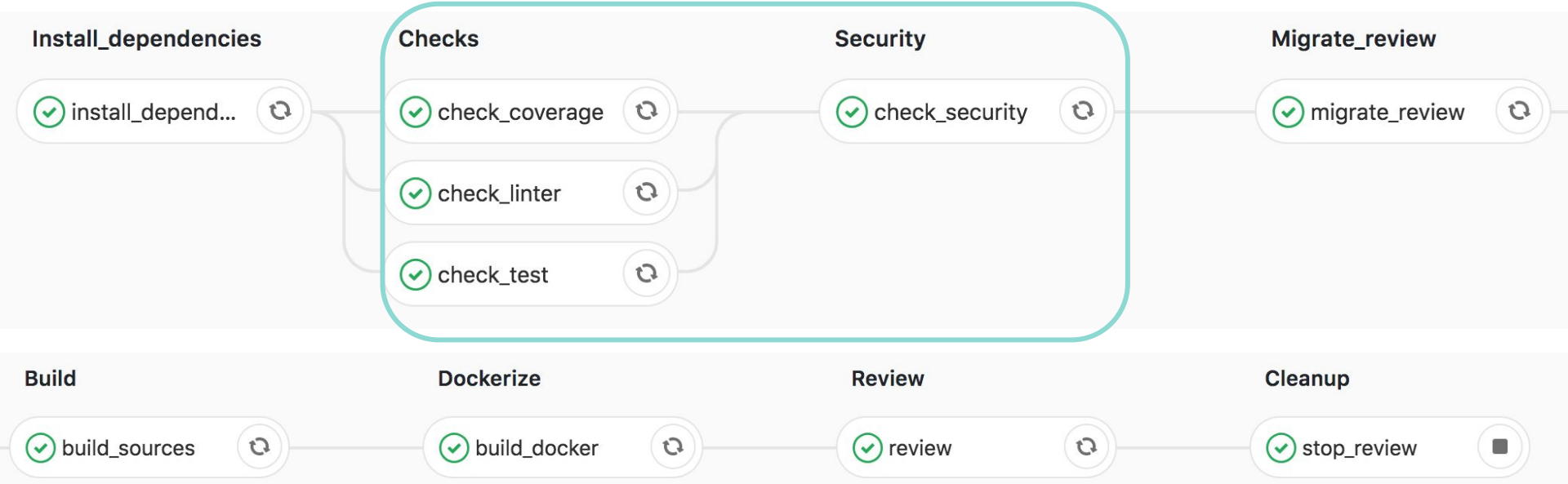
Pipeline definition - Pull Request Stages



Pipeline definition - Pull Request Stages

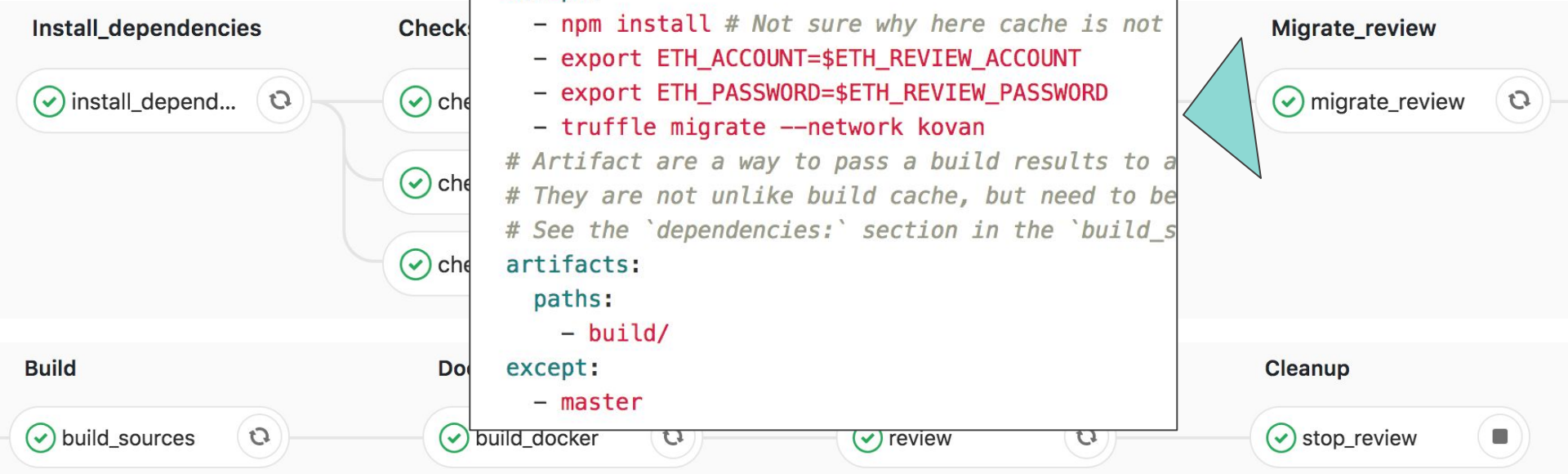


Pipeline definition - Pull Request Stages

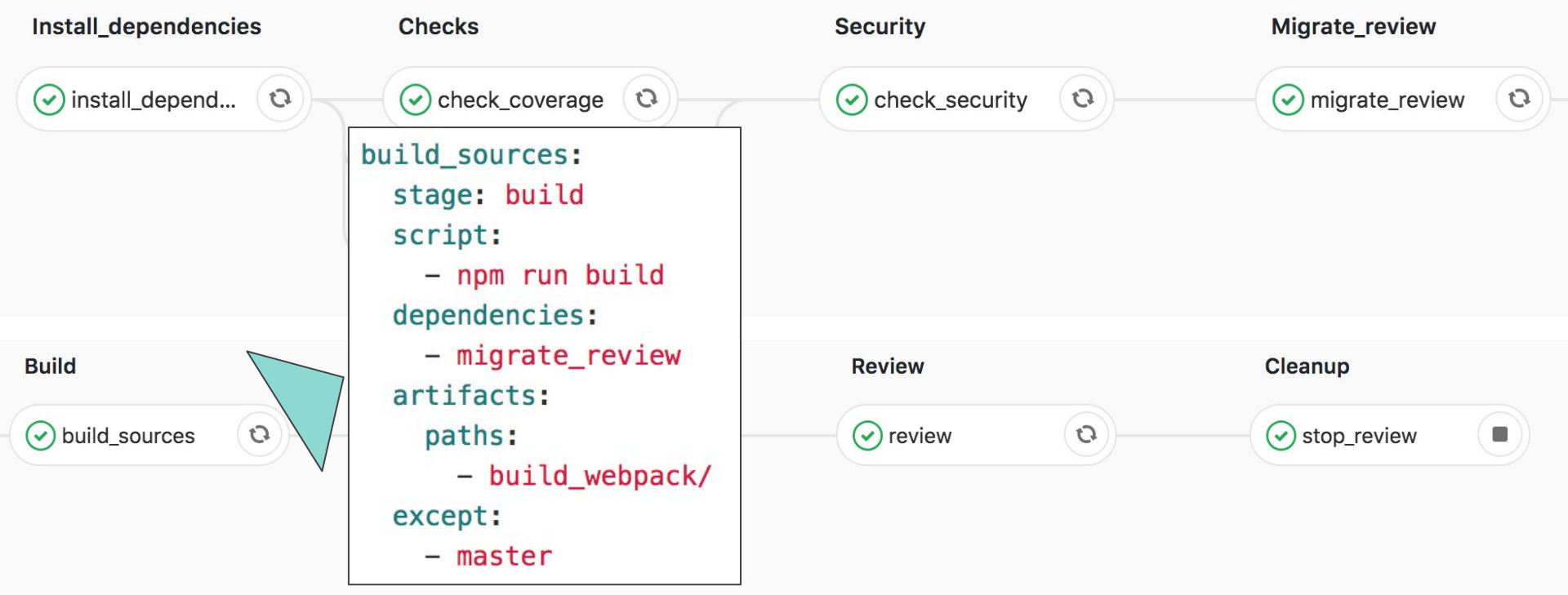


Pipeline t Stages

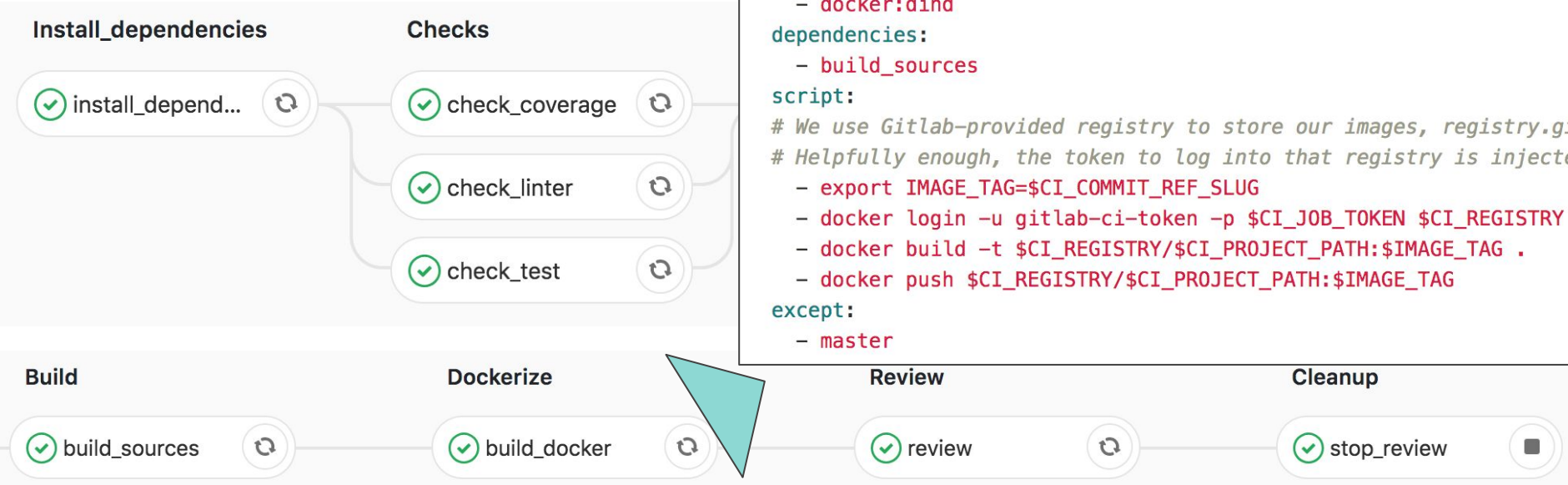
```yaml
migrate_review:
  stage: migrate_review
  variables:
    PROVIDER_ENDPOINT: "https://kovan.ethberl.in"
  script:
    - npm install # Not sure why here cache is not
    - export ETH_ACCOUNT=$ETH_REVIEW_ACCOUNT
    - export ETH_PASSWORD=$ETH_REVIEW_PASSWORD
    - truffle migrate --network kovan
  # Artifact are a way to pass a build results to a
  # They are not unlike build cache, but need to be
  # See the `dependencies:` section in the `build_s
  artifacts:
    paths:
      - build/
  except:
    - master
```



Pipeline definition - Pull Request Stages



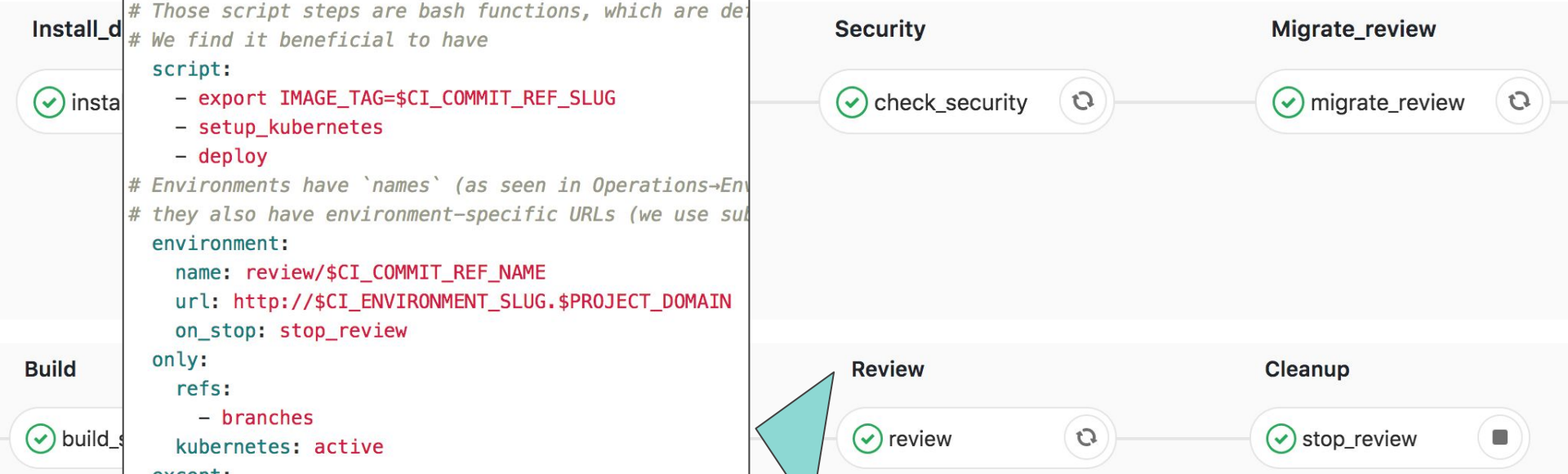
Pipeline definitio




```yaml
build_docker:
  stage: dockerize
  image: docker:stable # Build step with a non-default docker image
  cache: {} # We don't need npm cache here
  # `dind` stands for Docker-IN-Docker. While `docker:stable` image p
  # `dind` image serves as a Docker **daemon**. (And `dind` is set up
  services:
    - docker:dind
  dependencies:
    - build_sources
  script:
  # We use Gitlab-provided registry to store our images, registry.git
  # Helpfully enough, the token to log into that registry is injected
    - export IMAGE_TAG=$CI_COMMIT_REF_SLUG
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
    - docker build -t $CI_REGISTRY/$CI_PROJECT_PATH:$IMAGE_TAG .
    - docker push $CI_REGISTRY/$CI_PROJECT_PATH:$IMAGE_TAG
  except:
    - master
```

n - Pull Request Stages

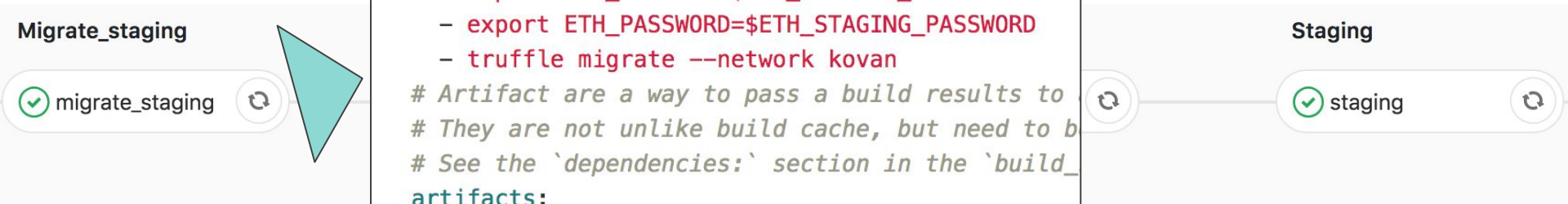
```
review:
  stage: review
  image: roffe/kubectl
  cache: {} # We don't need npm cache here
# Those script steps are bash functions, which are de
# We find it beneficial to have
  script:
    - export IMAGE_TAG=$CI_COMMIT_REF_SLUG
    - setup_kubernetes
    - deploy
# Environments have `names` (as seen in Operations→Env
# they also have environment-specific URLs (we use su
  environment:
    name: review/$CI_COMMIT_REF_NAME
    url: http://$CI_ENVIRONMENT_SLUG.$PROJECT_DOMAIN
    on_stop: stop_review
  only:
    refs:
      - branches
    kubernetes: active
  except:
    - master
```



Pipeline definition - Post-PR Stages

Pipe Stages

```
migrate_staging:
  stage: migrate_staging
  variables:
    PROVIDER_ENDPOINT: "https://kovan.ethberl.in"
  script:
    - npm install # Not sure why here cache is not
    - export ETH_ACCOUNT=$ETH_STAGING_ACCOUNT
    - export ETH_PASSWORD=$ETH_STAGING_PASSWORD
    - truffle migrate --network kovan
  # Artifact are a way to pass a build results to
  # They are not unlike build cache, but need to b
  # See the `dependencies:` section in the `build_
  artifacts:
    paths:
      - build/
  only:
    refs:
      - master
```

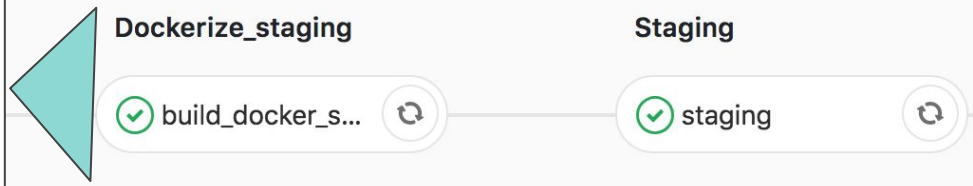


Pipeline definition


```
build_sources_staging:
  stage: build_staging
  script:
    - npm run build
  dependencies:
    - migrate_staging
  artifacts:
    paths:
      - build_webpack/
  only:
    refs:
      - master
```

Pipeline definition - Post-PR Stages

```yaml
build_docker_staging:
  stage: dockerize_staging
  image: docker:stable # Build step with a non-default docker image
  cache: {} # We don't need npm cache here
  # `dind` stands for Docker-IN-Docker. While `docker:stable` image p
  # `dind` image serves as a Docker **daemon**. (And `dind` is set up
  services:
    - docker:dind
  dependencies:
    - build_sources_staging
  script:
  # We use Gitlab-provided registry to store our images, registry.git
  # Helpfully enough, the token to log into that registry is injected
    - export IMAGE_TAG=${CI_COMMIT_SHA:0:8}staging
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
    - docker build -t $CI_REGISTRY/$CI_PROJECT_PATH:$IMAGE_TAG .
    - docker push $CI_REGISTRY/$CI_PROJECT_PATH:$IMAGE_TAG
  only:
    refs:
      - master
```



Pipeline definition - Post-PR Stages

```
staging:
  stage: staging
  image: roffe/kubectl
  cache: {} # We don't need npm cache here
  script:
    - export IMAGE_TAG=${CI_COMMIT_SHA:0:8}staging
    - setup_kubernetes
    - deploy
  environment:
    name: staging
    url: http://staging.$PROJECT_DOMAIN
  only:
    refs:
      - master
    kubernetes: active
```

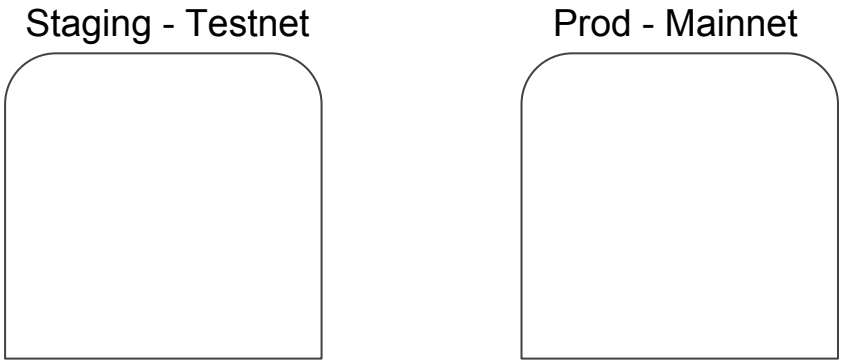
Contract Iterations + Artifact Publishing

- Truffle migrate artifact build file
- Migration Addresses + ABI
- Migration status
    - Only migrate contracts that have updated deltas

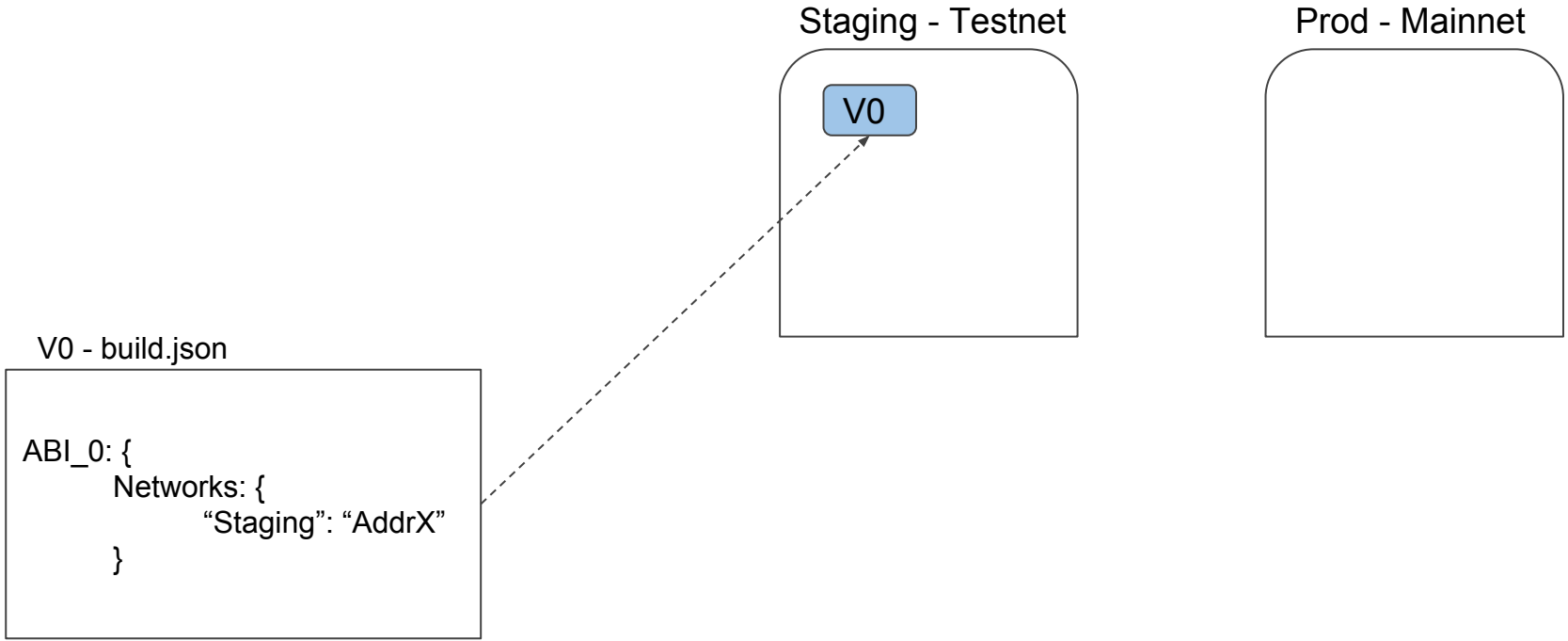
Contract Versioning

- BuildContract.json file is updated on every migration for each target network
  - Contains same ABI but different address per network
- Use Release Tags
  - When deployment to your production environment create release tag

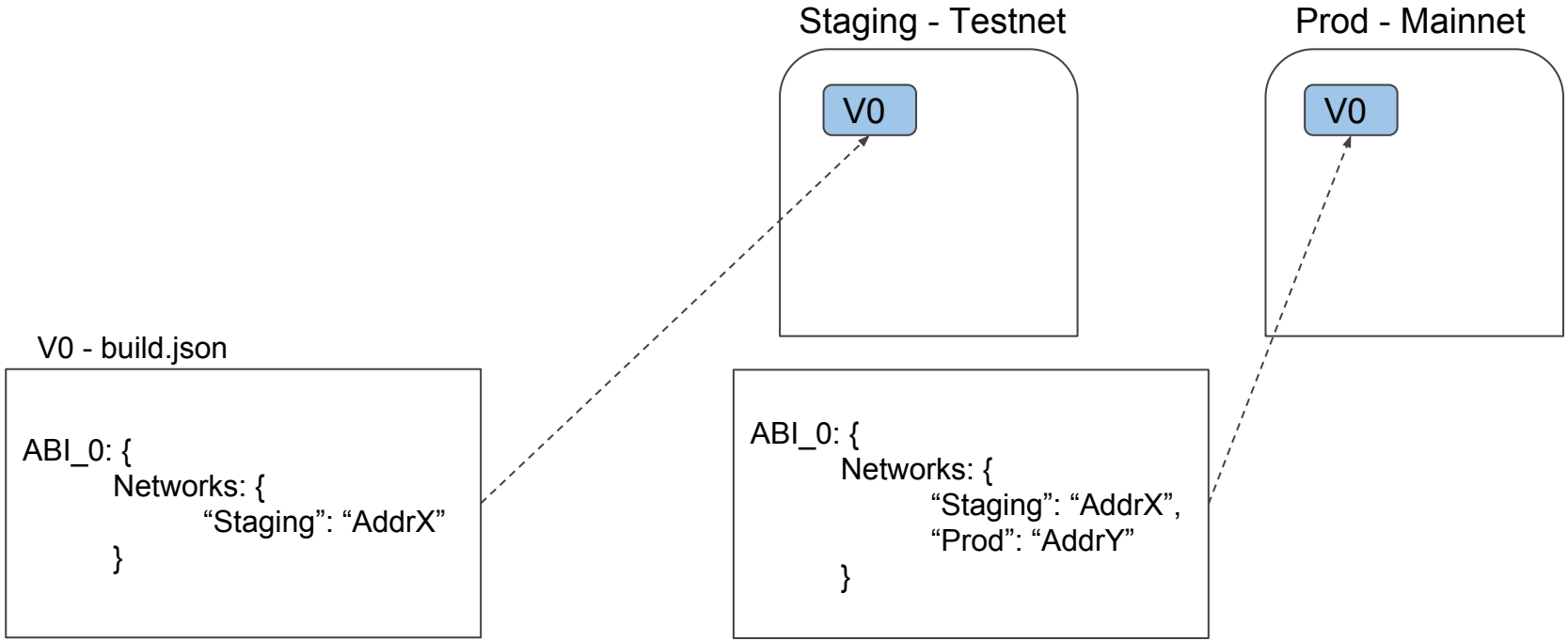
Contract Versioning - Flow Diagram



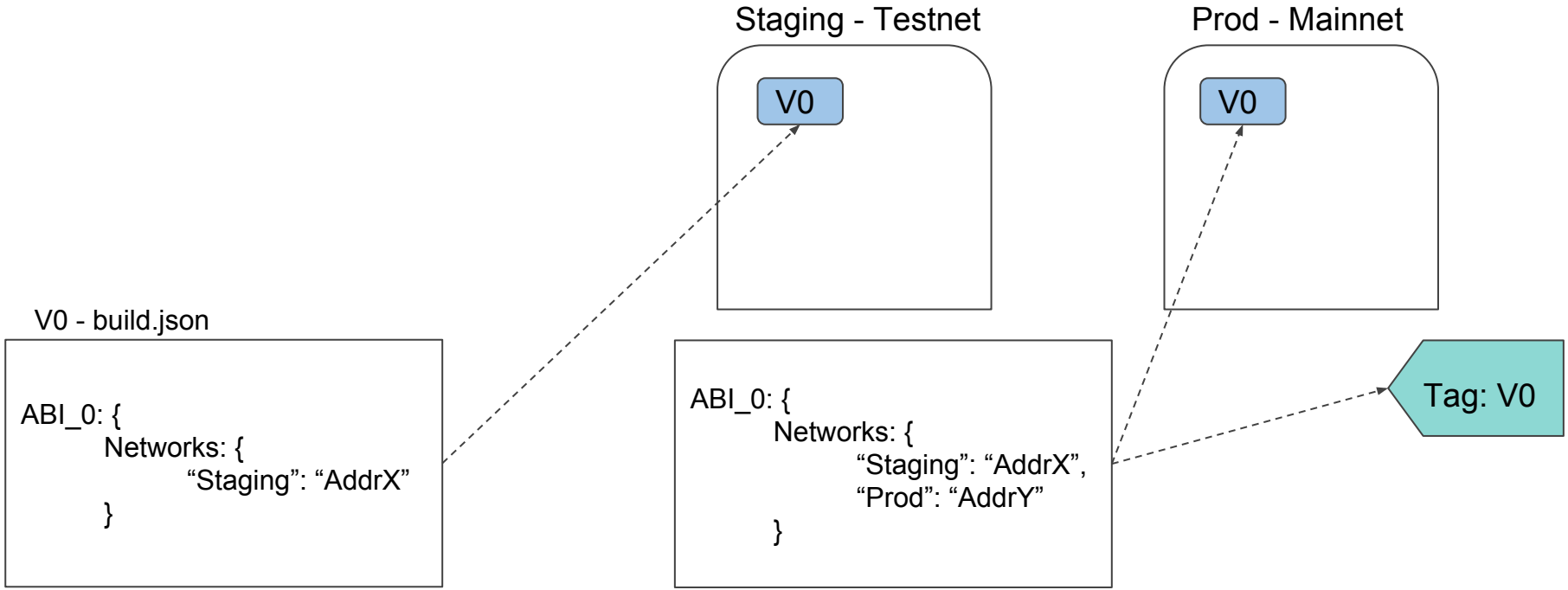
Contract Versioning - Flow Diagram



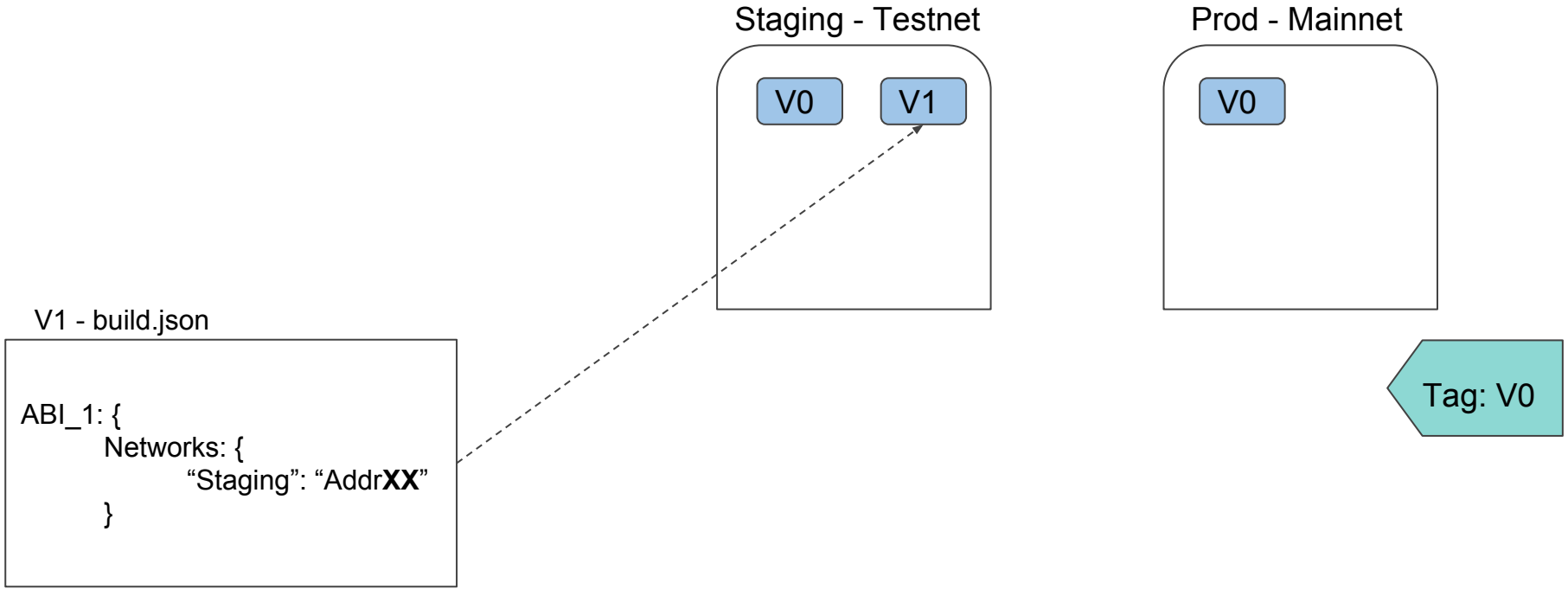
Contract Versioning - Flow Diagram



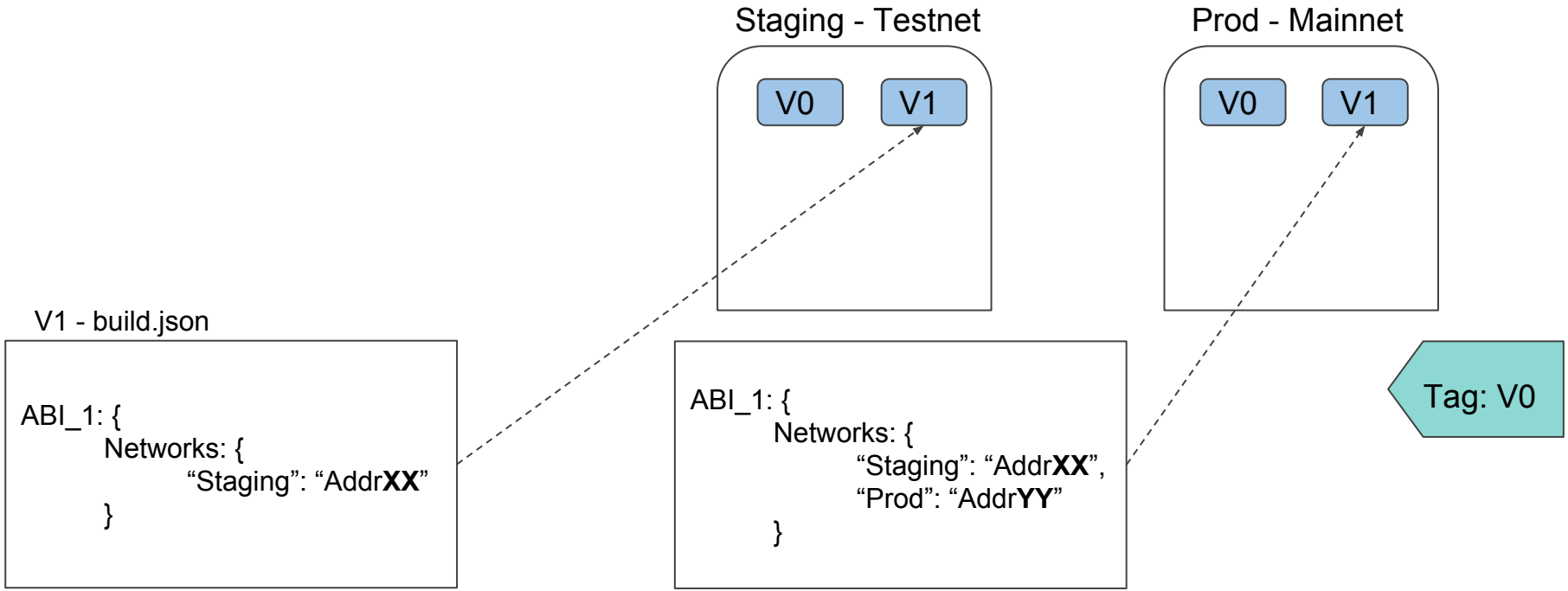
Contract Versioning - Flow Diagram



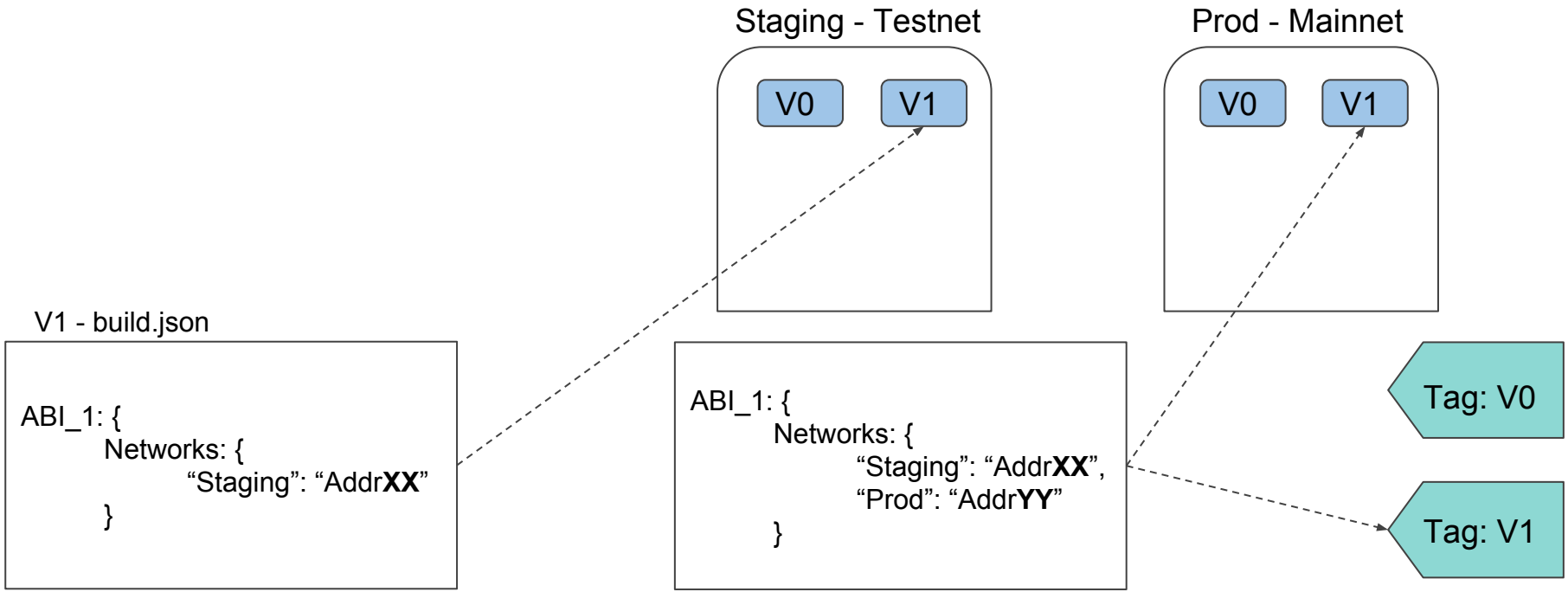
Contract Versioning - Flow Diagram



Contract Versioning - Flow Diagram



Contract Versioning - Flow Diagram



Key Management per environment

- Review test network: gitlabci or k8s worker preconfigured secrets
- Testnet Rinkeby/Kovan/...: gitlabci or k8s worker preconfigured secrets
- Mainnet: Manual migration

Demo Code

ETHBerlin CI/CD Template Code:

https://github.com/ethberlin-hackathon/ETHBerlin-KnowledgeBase/blob/master/hackers.md

https://gitlab.com/mikiquantum/simple-dapp-calculator

Libraries:

https://github.com/ConsenSys/mythril

https://github.com/sc-forks/solidity-coverage

https://github.com/protofire/solhint

Thank you, and Happy Hacking!