RAI for Administrators

10/27/17

Introduction

The following document explains what rai is, how to set it up, and how to configure it.

Terminology

We will use the following terms throughout this document. For the publically available services, this document assume reader is familiar with them, their limitations, and semantics:

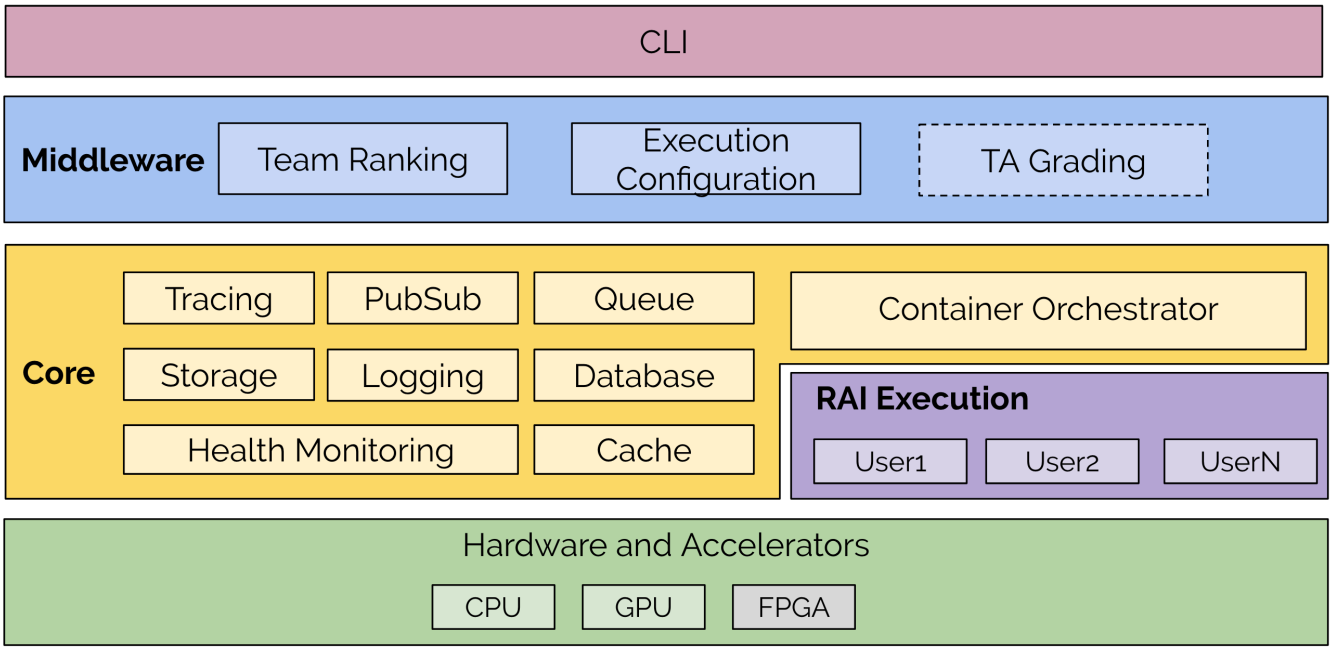
- **Client/rai**: the client which users use to interact with the RAI system. This includes the RAI client as well as the docker builder website. The client is usually installed on user's machine and is used primarily to submit tasks to the system. At any point in time, there can be more than

one client running and submitting jobs to the system. The client should work on any OS and does not require any special hardware to be installed on the system.

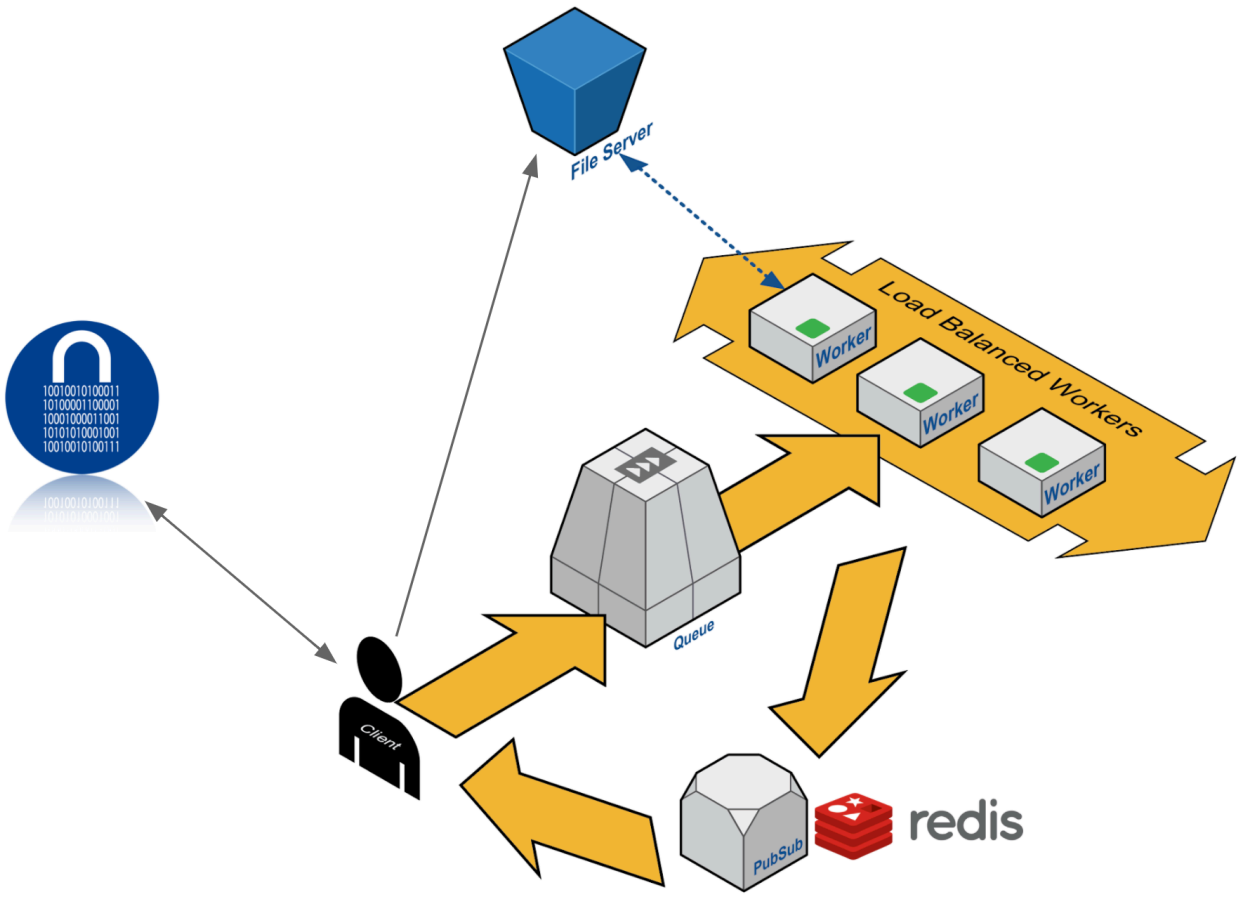
- **Job Queue**: User's jobs are submitted onto a queue (using sharding, for example). The queue currently uses Amazon's SQS queue system. There can be more than one queue, but we currently use only one.
- **Pub/Sub Queue**: Output for jobs are published onto a pub/sub server. RAI currently uses Redis for pub/sub. Multiple redis servers can be used, but we currently use only one.
- **Server/raid**: All work/execution is run on the server. The server listens to the queue and executes jobs if capable. Depending on the load, more than one server can be run at any point in time. The number of jobs that a server can handle is configurable. Linux is required to use the server with GPU capabilities.
- **Docker**: User code execution occurs only within a docker container. Docker is also used to build docker images as well as publishing images to the docker registry.
- **CUDA Volume**: A docker plugin that mounts nvidia driver and cuda libraries within the launched docker container.
- **STS**: Amazon's STS allows one to place a time constraints on the amazon access tokens (also known as roles) being issued. The current constraint is 15 minutes.
- **Store/File Server**: The location where user's data files are stored. The rai system currently uses Amazon's S3 for storage.
- **Auth**: Only users with credentials can submit tasks to rai. Authentication keys can be generated using the **rai-keygen** tool. In the backend, we use Auth0 as the database.
- **App secret**: all keys, such as credentials to login to the pub/sub server, are encrypted using AES32 based encryption. For prebuilt binaries, the app secret is embedded in the executable. A user can specify the secret from the command line as well.

Note: I did ask for one to investigate Hashicorp's Vault to avoid us from using Auth0.

Components


CLI
Middleware
Team Ranking
Execution Configuration
TA Grading
Core
Tracing
PubSub
Queue
Storage
Logging
Database
Health Monitoring
Cache
Container Orchestrator
RAI Execution
User1
User2
UserN
Hardware and Accelerators
CPU
GPU
FPGA

Execution Flow



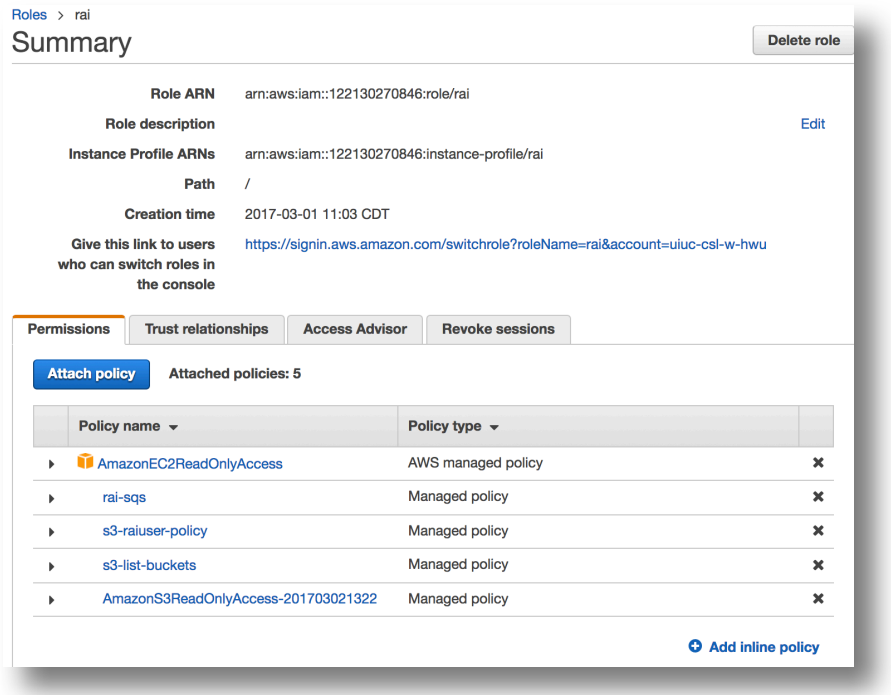
1. A client submits a task to RAI
    a. Credentials are validated
    b. The directory is archived and uploaded to the file server
    c. The task is submitted to the queue
    d. The user subscribes to the pub/sub server and prints all messages being received
2. A server accepts a task from the queue
    a. Check if the task is valid
    b. Either build or downloads the docker images required to run the task
    c. Download the user directory
    d. Start a publish channel to the pub/sub server

e. Start a docker container and run the user commands (if gpu is requested, then the cuda volume is mounted)
f. All stdout/stderr messages from the docker container are forwarded to the publish channel
g. Wait for either tasks to complete or a timeout
h. The output directory is uploaded to the file server and a link is published
i. Close the publish channel / docker container

Prerequisites

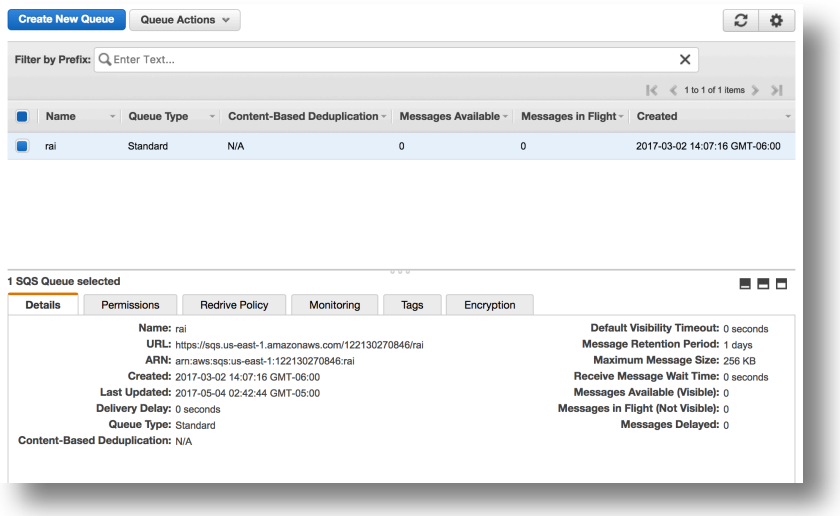
STS Permissions

Setup STS permissions so that users can upload to an S3 bucket and publish to an SQS queue. This can be done via the IAM AWS console. The STS currently being used only allows one to attach to a rai role which has very limited permissions:



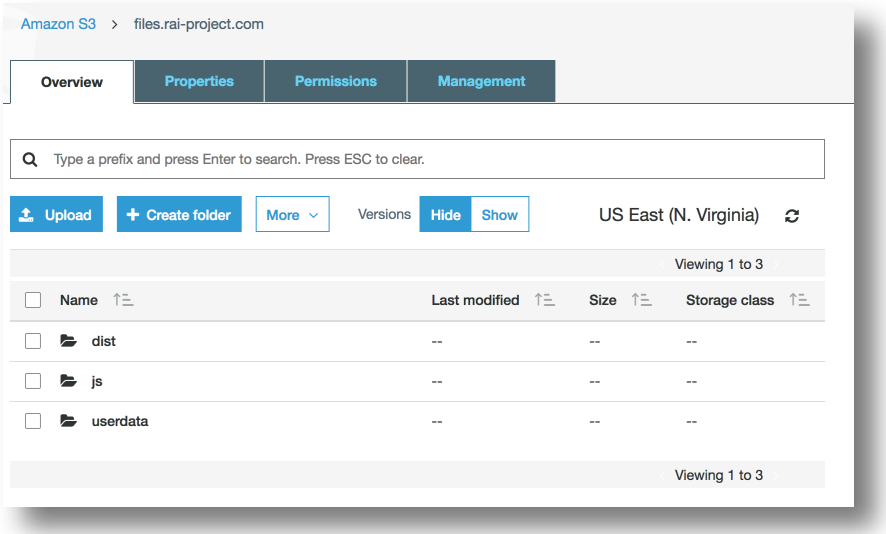
SQS Queue

Create an SQS queue using the AWS console



S3 Bucket

Create an S3 bucket using the AWS console



Redis Server

Install a redis server. A docker container can also be used.

CUDA Volume Plugin

Follow the instruction at https://github.com/rai-project/rai-docker-volume#rai-docker-volume- .

Prebuilt binaries exist on s3 in /files.rai-project.com/dist/rai-docker-volume/stable/latest

Note: These binaries are not publically readable and you need an AWS_KEY / SECRET to access them.

RAI Client Installation

See https://github.com/rai-project/rai#download-binaries

RAID Server Installation From Binary

Prebuilt raid binaries exist on s3 in /files.rai-project.com/dist/raid/stable/latest

Note: These binaries are not publically readable and you need an AWS_KEY / SECRET to access them.

RAID Server Installation From Source

1. Install golang. Either use the Go Version Manager or navigate to the Golang Site and set it up manually. It is preferred that you use the Go version manager.
2. Install glide. Follow the directions in https://github.com/Masterminds/glide#install
3. Clone the raid repository

```
mkdir -p $GOPATH/src/github.com/rai-project
cd $GOPATH/src/github.com/rai-project
git clone git@github.com:rai-project/raid.git
```

4. Install the software dependencies using glide

```
cd raid
glide install
```

5. Create an executable

```
go build
```

6. To embed the secret key into the executable, you will need to pass in a link flag

```
go build –ldflags="-s -w -X main.AppSecret=${APP_SECRET}"
```

7. Consult the travis file (https://github.com/rai-project/raid/blob/master/.travis.yml) for more build options

---

Configuration

Much of rai/raid is controlled by configuration files. Services that are shared between the client and server must match. In this section we will explain the minimal configurations needed for both the client and server

Note: One can create secret keys recognizable by rai/raid using rai-crypto[1] tool. If you want to encrypt a string using "PASS" as your app secret, then you'd want to invoke

```
rai-crypto encrypt -s PASS MY_PLAIN_TEXT_STRING
```

Configurations are specified in YAML format and exist either in **`$HOME/.rai_config.yml`** or are embedded within the executable. There are many more configurations that can be set, but if omitted then sensible defaults are used.

[1] https://github.com/rai-project/utils/tree/master/rai-crypto

Client Configuration

The client configuration configures the client for usage with a cluster of rai servers.

```
app:
  name: rai # name of the application
  verbose: false # whether to enable verbosity by default
  debug: false # whether to enable debug output by default
aws:
  access_key_id: AWS_ACCESS_KEY # the aws access key (encrypted)
  secret_access_key: AWS_SECRET_KEY # the aws secret key
  region: us-east-1 # the aws region to use
  sts_account: STS_ACCOUNT # the sts account number
  sts_role: rai # the sts role
  sts_role_duration_seconds: 15m # the maximum time period the sts role can be assumed
store:
  provider: s3 # the store provider
  base_url: http://s3.amazonaws.com # the base url of the file store
  acl: public-read # the default permissions for the files uploaded to the file store
client:
  name: rai # name of the client
  upload_bucket: files.rai-project.com # base url or the store buceket
  bucket: userdata # location to store the uploaded user data (user input)
  build_file: rai_build # location to store the result build data (user output)
auth:
  provider: auth0 # the authentication provider
  domain: raiproject.auth0.com # the domain of the authentication provider
  client_id: AUTH0_CLIENT_ID # the client id from the authentication. The auth0 client id
for example
  client_secret: AUTH0_CLIENT_SECRET # the client secret from the authentication. The auth0
client secret for example
pubsub:
  endpoints: # list of endpoints for the pub sub service
    - pubsub.rai-project.com:6379 # the pubsub server location + port
  password: PUBSUB_PASSWORD # password to the pub/sub service
```

Note: During the travis build process the client configurations are embedded into the client binary. Therefore the **$HOME/.rai_config.yml** is never read.

Server Configuration

All servers within a cluster share the same configuration. Here is the configuration currently being used:

```
app:
  name: rai # name of the application
  verbose: false # whether to enable verbosity by default
  debug: false # whether to enable debug output by default
```

```
logger:
  hooks: # log hooks to enable
    - syslog # enable the syslog hook
aws:
  access_key_id: AWS_ACCESS_KEY # the aws access key (encrypted)
  secret_access_key: AWS_SECRET_KEY # the aws secret key
  region: us-east-1 # the aws region to use
store:
  provider: s3 # the store provider
  base_url: http://s3.amazonaws.com # the base url of the file store
  acl: public-read # the default permissions for the files uploaded to the file store
  acl: public-read
broker: # broker/queue configuration section
  provider: sqs # the queue provider
  serializer: json # the serialization method to use for messages. Json is the default
  autoack: true # enable auto acknowledgement of messages
store:
  provider: s3 # the store provider
  base_url: http://s3.amazonaws.com # the base url of the file store
  acl: public-read # the default permissions for the files uploaded to the file store
client:
  name: rai # name of the client
  upload_bucket: files.rai-project.com # base url or the store buceket
  bucket: userdata # location to store the uploaded user data (user input)
  build_file: rai_build # location to store the result build data (user output)
auth:
  provider: auth0 # the authentication provider
  domain: raiproject.auth0.com # the domain of the authentication provider
  client_id: AUTH0_CLIENT_ID # the client id from the authentication. The auth0 client id
for example
  client_secret: AUTH0_CLIENT_SECRET # the client secret from the authentication. The auth0
client secret for example
pubsub:
  endpoints: # list of endpoints for the pub sub service
    - pubsub.rai-project.com:6379 # the pubsub server location + port
  password: PUBSUB_PASSWORD # password to the pub/sub service
```

Other useful configuration options are docker.time_limit (default 1 hour), docker.memory_limit (default 16gb)

Start / Stop Server

With the absence of integration of **raid** with system service management (such as SystemD, UpStart, …) one needs to start and stop the **raid** server manually. Assuming you've already compiled the **raid** executable, you can run the server using the following command:

```
./raid -s ${MY_SECRET}
```

There are a few options that are available to control settings:

Table 1 : Command line options for raid server

Debug Mode	-d
Verbose Mode	-v
Application Secret	-s MY_SECRET

Note: I did ask for one to create a SystemD service that would simplify this process. Carl started this process in May/June, but it was never complete. I also had this on my list of tasks to be complete to simplify the management of RAI.

The above command will exit when a user exists the terminal session. Use the nohup command to avoid that

```
nohup ./raid -d -v -s ${MY_SECRET} &
```

Creating RAI Accounts

Either build the rai-keygen or download the prebuilt binaries which exist on s3 in /files.rai-project.com/dist/rai-keygen/stable/latest

Note: These binaries are not publically readable and you need an AWS_KEY / SECRET to access them.

One can use the rai-keygen https://github.com/rai-project/rai-keygen/blob/master/README.md to generate RAI and email account information to the people enrolled in the class. The mailing process

uses mailgun. A prebuilt rai-keygen includes a builtin configuration file, but if compiling from source, then you need to add the email configuration options

```
email:
  provider: mailgun # the email provider
  domain: email.webgpu.com # the domain of the
  source: postmaster@webgpu.com # the source email
  mailgun_active_api_key: API_KEY
  mailgun_email_validation_key: VALIDATION_KEY
```

You will not need the above if you do not need to email the generated keys.

Note: Docker builder does not require account generation, since account information is embedded into the webserver.

Administration Tips

The following tips are based on past experience administering clusters and managing arbitrary user execution:

- If using CUDA, make sure to enable persistence mode http://docs.nvidia.com/deploy/driver-persistence/index.html . A systemd service exists within the raid repository
- A system can become unstable when executing arbitrary code. Consult the logs (ideally a distributed logging) when trying to identify why certain tasks succeed while other fail.
- Install Cadvistor (github.com/google/cadvisor) to examine the health of the docker container and monitor them.
- Make sure that you have enough disk space. For example, last year the redis server ran out of disk space 2-3 days before the deadline.