Ctrl

ctrl is a Command & Control (C2) backend system for Servers, IOT and Edge platforms. Simply put, control anything that runs an operating system.

Example use cases:

- Send shell commands or scripts to one or many end nodes that will instruct to change config, restart services and control those systems.
- Gather data from both secure and not secure devices and systems, and transfer them encrypted in a secure way over the internet to your central system for handling those data.
- Collect metrics or monitor end nodes, then send and store the result to some ctrl instance, or pass those data's on to another ctrl instance for further handling of metrics or monitoring data.
- Distribute certificates.
- Run as a sidecar in Kubernetes for direct access to the pod.

As long as you can do something as an operator in a shell on a system you can do the same with ctrl in a secure and encrypted way to one or all end nodes (servers) in one go with one single message/command.

Features

- Run bash commands or complete scripts of your prefered scripting language (bash, python, powershell and so on).
- Read and write to files.
- Copy files.
- ACL's to restric who can do what.

Example

An example of a **request** message to copy into ctrl's **readfolder**.

Quick start

Start up a local nats message broker

```
docker run -p 4444:4444 nats -p 4444
```

Create a ctrl docker image.

```
git clone git@github.com:postmannen/ctrl.git
cd ctrl
docker build -t ctrl:test1 .
mkdir -p testrun/readfolder
cd testrun
```

create a .env file

```
cat << EOF > .env
NODE_NAME="node1"
BROKER_ADDRESS="127.0.0,1:4444"
ENABLE_DEBUG=1
START_PUB_REQ_HELLO=60
IS_CENTRAL_ERROR_LOGGER=0
EOF
```

Start the ctrl container.

```
docker run --env-file=".env" --rm -ti -v $(PWD)/readfolder:/app/readfolder ctrl:test1
```

Send a message.

```
cat << EOF > msg.yaml
---
- toNodes:
    - node1
  method: REQCliCommand
  methodArgs:
    - "bash"
    - "-c"
    - |
      echo "some config line" > /etc/my-service-config.1
      echo "some config line" > /etc/my-service-config.2
      echo "some config line" > /etc/my-service-config.3
      systemctl restart my-service

  replyMethod: REQNone
  ACKTimeout: 0
EOF

cp msg.yaml readfolder
```

Input methods

New Request Messages in Json/Yaml format can be injected by the user to ctrl in the following ways:

- **Unix Socket**. Use for example netcat or another tool to deliver new messages to a socket like `nc -U tmp/ctrl.sock < msg.yaml` .
- **Read Folder**. Write/Copy messages to be delivered to the `readfolder` of ctrl.
- **TCP Listener**, Use for example netcat or another tool to deliver new messages a TCP Listener like `nc localhost:8888 < msg.yaml` .

Error messages from nodes

- Error messages will be sent back to the central error handler and the originating node upon failure.

The error logs can be read on the central server in the directory `<ctrl-home>/data/errorLog` , and in the log of the instance the source node. You can also create a message to read the errorlog if you don't have direct access to the central server.

Flags and configuration file

ctrl supports both the use of flags with env variables. An .env file can also be used.

Schema for the messages to send into ctrl via the API's

Name	value	description
toNode	`string`	A single node to send a message to
toNodes	`string array`	A comma separated list of nodes to send a message to
method	`string`	The request method to use
methodArgs	`string array`	The arguments to use for the method
replyMethod	`string`	The method to use for the reply message
replyMethodArgs	`string array`	The arguments to use for the reply method
ACKTimeout	`int`	The time to wait for a received acknowledge (ACK). 0 for no acknowledge
retries	`int`	The number of times to retry if no ACK was received
replyACKTimeout	`int`	The timeout to wait for an ACK message before we retry
replyRetries	`int`	The number of times to retry if no ACK was received for repply messages
methodTimeout	`int`	The timeout in seconds for how long we wait for a method to complete

Name	value	description
replyMethodTimeout	`int`	The timeout in seconds for how long we wait for a method to complete for repply messages
directory	`string`	The directory for where to store the data of the repply message
fileName	`string`	The name of the file for where we store the data of the reply message
schedule	[int type value for interval in seconds, int type value for total run time in seconds]	Schedule a message to re run at interval

Request Methods

Method name	Description
REQOpProcessList	Get a list of the running processes
REQOpProcessStart	Start up a process
REQOpProcessStop	Stop a process
REQCliCommand	Will run the command given, and return the stdout output of the command when the command is done
REQCliCommandCont	Will run the command given, and return the stdout output of the command continously while the command runs
REQTailFile	Tail log files on some node, and get the result for each new line read sent back in a reply message
REQHttpGet	Scrape web url, and get the html sent back in a reply message
REQHello	Send Hello messages
REQCopySrc	Copy a file from one node to another node
REQErrorLog	Method for receiving error logs for Central error logger
REQNone	Don't send a reply message

Method name	Description
REQToConsole	Print to stdout or stderr
REQToFileAppend	Append to file, can also write to unix sockets
REQToFile	Write to file, can also write to unix sockets

History

ctrl is the continuation of the code I earlier wrote for RaaLabs called Steward. The original repo was public with a MIT license, but in October 2023 the original repo was made private, and are no longer avaialable to the public. The goal of this repo is to provide an actively maintained, reliable and stable version. This is also a playground for myself to test out ideas an features for such a service as described earlier.

This started out as an idea I had for how to control infrastructure. This is the continuation of the same idea, and a project I'm working on free of charge in my own spare time, so please be gentle 😃

NB: Filing of issues and bug fixes are highly appreciated. Feature requests will genereally not be followed up simply because I don't have the time to review it at this time :

Steward was written with an MIT License. With the new fork the service was renamed to ctrl and the license were changed to AGPL V3.0. More information in the LICENSE-CHANGE.md and LICENSE files.

Disclaimer

All code in this repository are to be concidered not-production-ready, and the use is at your own responsibility and risk. The code are the attempt to concretize the idea of a purely async management system where the controlling unit is decoupled from the receiving unit, and that that we know the state of all the receiving units at all times.

Also read the license file for further details.

Expect the main branch to have breaking changes. If stability is needed, use the released packages, and read the release notes where changes will be explained.