Viking Configuration Management System

Adrian Hungate

Jul 10, 2023

CONTENTS

Note: This project is under active development.

Viking CMS is a client/server system. The server runs on one (or more) central machine from which the Viking Agents can request the details of your Longship, which includes your Hammars and Hammar Strikes and multi-tiered rigging (named settings) which each Strike can use.

When an Agent requests its portion of the Longship, it will get back a list of Hammars it needs to download, including their version numbers. This allows Agents to cache Hammar content locally (It shouldn't change too often, even if your configuration changes), optimising download times.

The Viking Server does not access or run Hammar Strikes, instead all interpretation of the Longship, and Hammar Strikes is done on the Agent. Most machines running VikingServer will, however, also run VikingAgent, to manage their own configuration.

A single run of a Longship is called a Voyage.

CHAPTER

ONE

CONTENTS

1.1 Viking Agent

The Viking Agent is the workhorse of the Viking Configuration Management System. An instance of the Agent runs on every managed machine.

The Agent contains the parser and execution engine for interpreting Longships, downloading and caching Hammars, executing Strikes, and reporting back to the Viking Server with the results of the Voyage.

1.1.1 Agent Command Line Options

1.1.2 Voyage Sequence

1. Call For The Longship - Send a request, including my hostname, to the Viking Server for the Longship. This returns only sections relevant to this machine (i.e. Global and this host)
2. Pillage - Collection of information (spoils) about the machine that the agent is running on, including, but not limited to
    - Hostname (This is actually determined before calling for the longship, but is also added as a spoil)
    - CPU Type
    - CPU Core Count
    - Memory Quantity
    - Disk space per partition/mount point
3. Scan the longship for Hammars
    - For each Hammar
        - if that name/version is not already cached
            - Request if from the viking-forge and cache it (The longship can specify a local forge, and whether to try the master forge if the hammar is not found)
        - If that version is not already in the unpacked cache
            - Delete any existing unpacked files
            - Unpack the correct version
4. Execute the strikes in the Longship, in order, chronicling the Strike against that strike name in the Voyage Chronicle
5. Return the Voyage Chronicle

1.2 Agent API

The Agent API is the API provided by the Server that the Agent uses to download its Longship, its Hammars and to upload its results.

1.2.1 API Endpoints

Longship

- GET /api/v1/longship/<hostname>

Send

- Hostname in the path

Return

```
{
    "hammars": [
        "<hammarname>",
        "<hammarname>",
        ...
    ],
    "gobalrigging": {
        "<riggingname>": <riggingvalue>,
        ...
    },
    "localrigging": {
        "<riggingname>": <riggingvalue>,
        ...
    },
    "strikes": [
        "<strikename>": {
            "<riggingname>": <riggingvalue>,
            ...
        }
    ]
}
```

- variablevalue can be any valid JSON type

Hammar

- GET /api/v1/hammar/<hammarname>
- GET /api/v1/hammar/<hammarname>/<version>

Send

- Hostname in path
- Version in path (optional)

Return

- Hammar as a gzipped tar file

Chronicle

- POST /api/v1/chronicle/<hostname>

Send

- Hostname in path
- Report in body

```
{
    "date": "<datetime>",
    "hammars": [
        "<hammarname>",
        ...
    ],
    "strikes": [
        "<strikename>": {
            "starttime": "<datetime>",
            "endtime": "<datetime>",
            "log": [
                "<logline>",
                ...
            ]
        },
        ...
    ]
}
```

Return

```
{
    status: 'ok|rejected'
}
```

1.3 Hammar Script Syntax

Hammar Script is a Python compatible language with a few extensions to facilitate basic configuration management functions.

Each Hammar Script file should have a filename that ends in *.hs* and should live in the root of the Hammar. Empty Hammar Script files are not valid, each file must contain at least one strike.

1.3.1 Strikes

A strike is a Python function with the following signature:

```python
def strikename(longship: Viking.Longship) -> bool:
```

- The Longship object has the following additional attributes:
  - *spoils* - Values pillaged from the machine as the run started.
  - *globalrigging* - Globals from the longship.
- *strikename* - This must be unique within the Hammar.
- A strike should return True if it succeeded or False if it failed, other values will be evaluated in line with ECMAscript truth testing (e.g. None values, Empty strings, 0 etc will evaluate to False).
- Strikes should output using *longship.chant(message)* anything that they want included in the Longship Voyage Chronicle, the on-disk and on-screen log

Implicit includes

The namespace in which the Hammar Script executes has the modules of the Hammar Script Engine pre imported. Details of these classes and modules can be found in the class documentation.

Including python modules

Yeah, just don't try. I'm pretty certain I've fully disabled it. If you find a way of pulling in arbitrary Python modules please let me know, this is a security and consistency issue - All code should be part of the Viking library or Hammar Script.

Including Hammar Script

You can include the Strikes from another Hammar using

Which will present them as *hammarname . strikename*

Allowing one hammar to use and/or extend Strikes from another. For example a concat Hammar that allows a file to be built a piece at a time might use the file Hammar to provide basic file handling

1.4 Embedded Hammar Script

Hammar template files, located under */templates* in the templates, can contain elements of Embedded Hammar Script which is small sections of Hammar Script that either apply conditions to sections, produce direct output, or in some other way alter the file as it is being put in place.

Embedded Hammar Script is introduced by using EHS tokens like this:

```
# Config file
SETTING=5
{{EHS}}
if spoils['othersetting'] is not None:
print("setting=", EHS=spoils['othersetting'])
{{EHS-END}}
```

There is also a short syntax for inserting a spoils or variables:

```
The value of variable is {{variable}}
```