Stork

Release 0.9.0

Jul 01, 2020

Contents

`Stork` is a new project led by ISC with the aim of delivering an *ISC BIND 9* and *ISC Kea DHCP* use and monitoring dashboard. It is intended to be a spiritual successor of the earlier attempts *Kittiwake* and *Anterius*.

This is the reference guide for Stork version 0.9.0. Links to the most up-to-date version of this document, along with other documents for Stork, can be found on ISC's Stork project homepage or at readthedocs .

CHAPTER 1

Overview

1.1 Goals

The goals of the Stork project are as follows:

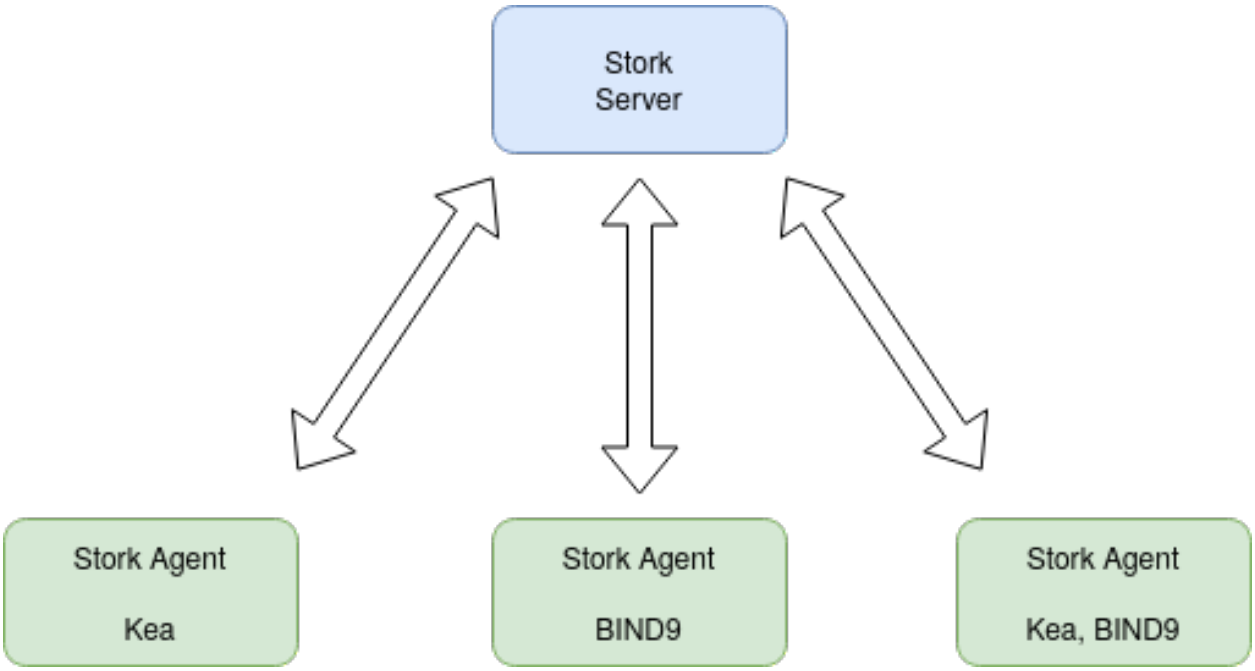
- to provide monitoring and insight into *ISC Kea DHCP* and *ISC BIND 9* operations
- to provide alerting mechanisms that indicate failures, fault conditions, and other unwanted events in *ISC Kea DHCP* and *ISC BIND 9* services
- to permit easier troubleshooting of these services

1.2 Architecture

Stork is comprised of two components: `Stork Server` and `Stork Agent`.

`Stork Agent` is installed along with *Kea DHCP* or *BIND 9* and interacts directly with those services. There may be many agents deployed in a network, one per machine.

`Stork Server` is installed on a stand-alone machine. It connects to any indicated agents and indirectly (via those agents) interacts with the *Kea DHCP* and *BIND 9* services. It provides an integrated, centralized front end for interacting with these services. Only one `Stork Server` is deployed in a network.


Stork
Server
Stork Agent
Kea
Stork Agent
BIND9
Stork Agent
Kea, BIND9

CHAPTER 2

Installation

Stork can be installed from pre-built packages or from sources. The following sections describe both methods.

2.1 Prerequisites

`Stork Server` and `Stork Agent` have been tested thoroughly on the Ubuntu 18.04 system. They have been tested and run on the Fedora 31 system as well.

The `Stork Agent` does not require any specific dependencies to run. It can be run immediately after installation.

Stork uses the *status-get* command to communicate with Kea, and therefore will only work with a version of Kea that supports *status'get*. The *status-get* command was introduced in Kea 1.7.3. At this time, Stork works with Kea version 1.7.3 and later versions only, although we intend to backport the *status-get* command to Kea 1.6.3.

Stork requires the premium Host Commands hook library to retrieve host reservations stored in an external database. Stork can retrieve host reservations stored locally in the Kea configuration without any additional hook libraries.

For the `Stork Server`, a PostgreSQL database (https://www.postgresql.org/) using at least version 11 of PostgreSQL is required. (The installation procedure for PostgreSQL is OS-specific and is not included here.)

These instructions prepare a database for use with the `Stork Server`, with the *stork* database user and *stork* password. Next, a database called *stork* is created and the *pgcrypto* extension is enabled in the database.

First, connect to PostgreSQL using *psql* and the *postgres* administration user:

```
$ psql postgres
psql (11.5)
Type "help" for help.
postgres=#
```

Then, prepare the database:

```
postgres=# CREATE USER stork WITH PASSWORD 'stork';
CREATE ROLE
postgres=# CREATE DATABASE stork;
```

(continues on next page)

(continued from previous page)

```
CREATE DATABASE
postgres=# GRANT ALL PRIVILEGES ON DATABASE stork TO stork;
GRANT
postgres=# \c stork
You are now connected to database "stork" as user "thomson".
stork=# create extension pgcrypto;
CREATE EXTENSION
```

2.2 Installing from Packages

Stork packages are stored in repositories located on the Cloudsmith service: https://cloudsmith.io/~isc/repos/stork/packages/. Both Debian/Ubuntu and RPM packages may be found there.

Detailed instructions for setting up the operating system to use this repository are available under the *Set Me Up* button on the Cloudsmith repository page.

2.2.1 Installing on Debian/Ubuntu

The first step for both Debian and Ubuntu is:

```
$ curl -1sLf 'https://dl.cloudsmith.io/public/isc/stork/cfg/setup/bash.deb.sh' | sudo␣
↪bash
```

Next, install the package with `Stork Server`:

```
$ sudo apt install isc-stork-server
```

Then, install `Stork Agent`:

```
$ sudo apt install isc-stork-agent
```

It is possible to install both agent and server on the same machine.

2.2.2 Installing on CentOS/RHEL/Fedora

The first step for RPM-based distributions is:

```
$ curl -1sLf 'https://dl.cloudsmith.io/public/isc/stork/cfg/setup/bash.rpm.sh' | sudo␣
↪bash
```

Next, install the package with `Stork Server`:

```
$ sudo dnf install isc-stork-server
```

Then, install `Stork Agent`:

```
$ sudo dnf install isc-stork-agent
```

It is possible to install both agent and server on the same machine.

2.2.3 Initial Setup of the Stork Server

These steps are the same for both Debian-based and RPM-based distributions that use *SystemD*.

After installing `Stork Server` from the package, the basic settings must be configured. They are stored in `/etc/stork/server.env`.

These are the required settings to connect with the database:

- STORK_DATABASE_HOST - the address of a PostgreSQL database; default is *localhost*
- STORK_DATABASE_PORT - the port of a PostgreSQL database; default is *5432*
- STORK_DATABASE_NAME - the name of a database; default is *stork*
- STORK_DATABASE_USER_NAME - the username for connecting to the database; default is *stork*
- STORK_DATABASE_PASSWORD - the password for the username connecting to the database

With those settings in place, the `Stork Server` service can be enabled and started:

```
$ sudo systemctl enable isc-stork-server
$ sudo systemctl start isc-stork-server
```

To check the status:

```
$ sudo systemctl status isc-stork-server
```

By default, the `Stork Server` web service is exposed on port 8080, so it can be visited in a web browser at http://localhost:8080.

It is possible to put `Stork Server` behind an HTTP reverse proxy using *Nginx* or *Apache*. In the `Stork Server` package an example configuration file is provided for *Nginx*, in */usr/share/stork/examples/nginx-stork.conf*.

2.2.4 Initial Setup of the Stork Agent

These steps are the same for both Debian-based and RPM-based distributions that use *SystemD*.

After installing `Stork Agent` from the package, the basic settings must be configured. They are stored in `/etc/stork/agent.env`.

These are the required settings to connect with the database:

- STORK_AGENT_ADDRESS - the IP address of the network interface which `Stork Agent` should use for listening for `Stork Server` incoming connections; default is *0.0.0.0* (i.e. listen on all interfaces)
- STORK_AGENT_PORT - the port that should be used for listening; default is *8080*

With those settings in place, the `Stork Agent` service can be enabled and started:

```
$ sudo systemctl enable isc-stork-server
$ sudo systemctl start isc-stork-server
```

To check the status:

```
$ sudo systemctl status isc-stork-server
```

After starting, the agent periodically tries to detect installed Kea DHCP or BIND 9 services on the system. If it finds them, they are reported to the `Stork Server` when it connects to the agent.

Further configuration and usage of the `Stork Server` and the `Stork Agent` are described in the *Using Stork* chapter.

2.3 Installing from Sources

2.3.1 Prerequisites

Stork sources can be built on Ubuntu 18.04 and Fedora 31.

There are two dependencies that need to be installed to build Stork sources:

- Rake
- Java Runtime Environment

Other dependencies are installed locally and automatically by Rake tasks.

For details about the environment, please see the Stork wiki at https://gitlab.isc.org/isc-projects/stork/wikis/Development-Environment.

2.3.2 Download Sources

The Stork sources are available on the ISC GitLab instance: https://gitlab.isc.org/isc-projects/stork.

To get the latest sources invoke:

```
$ git clone https://gitlab.isc.org/isc-projects/stork
```

2.3.3 Building

There are several components of Stork:

- Stork Agent - this is the binary *stork-agent*, written in Go
- Stork Server - this is comprised of two parts: - *backend service* - written in Go - *frontend* - an *Angular* application written in Typescript

All components can be built using the following command:

```
$ rake build_all
```

The agent component is installed using this command:

```
$ rake install_agent
```

and the server component with this command:

```
$ rake install_server
```

By default, all components are installed to the *root* folder in the current directory; however, this is not useful for installation in a production environment. It can be customized via the DESTDIR variable, e.g.:

```
$ sudo rake install_server DESTDIR=/usr
```

CHAPTER 3

Using Stork

This section describes how to use the features available in Stork. To connect to Stork, use a web browser and connect to port 8080. If Stork is running on a localhost, it can be reached by navigating to http://localhost:8080.

3.1 Managing Users

A default administrator account is created upon initial installation of Stork. It can be used to sign in to the system via the web UI, using the username admin and password admin.

To manage users, click on the Configuration menu and choose Users to see a list of existing users. There will be at least one user, admin.

To add a new user, click Create User Account. A new tab opens to specify the new account parameters. Some fields have specific restrictions:

- Username can consist of only letters, numbers, and an underscore (_).
- The e-mail field is optional, but if specified, it must be a well-formed e-mail.
- The firstname and lastname fields are mandatory.
- The password must only contain letters, digits, @, ., !, +, or -, and must be at least eight characters long.

Currently, users are associated with one of the two predefined groups (roles), i.e. super-admin or admin, which must be selected when the user account is created. Users belonging to the super-admin group are granted full privileges in the system, including creation and management of user accounts. The admin group has similar privileges, except that the users in this group are not allowed to manage other users' accounts.

Once the new user account information has been specified and all requirements are met, the Save button becomes active and the new account can be enabled.

3.2 Changing a User Password

An initial password is assigned by the administrator when a user account is created. Each user should change the password when first logging into the system. To change the password, click on the `Profile` menu and choose `Settings` to display the user profile information. Click on `Change password` in the menu bar on the left and specify the current password in the first input box. The new password must be specified and confirmed in the second and third input boxes, and must meet the password requirements specified in the previous section. When all entered data is valid, the `Save` button is activated for changing the password.

3.3 Deploying Stork Agent

The Stork system uses agents to monitor services. `Stork Agent` is a daemon that must be deployed and run on each machine to be monitored. Currently, there are no automated deployment routines and `Stork Agent` must be installed manually. This can be done in one of two ways: from RPM or deb packages (described in the *Installation* chapter), or by simply copying the `Stork Agent` binary to the destination machine manually.

Assuming services will be monitored on a machine with the IP 192.0.2.1, enter the following on the Stork server command line:

```
$ cd <stork-dir>
$ scp backend/cmd/stork-agent login@192.0.2.1:/path
```

On the machine to be monitored, start the agent by running:

```
$ ./stork-agent
```

It is possible to set the `--host=` or `STORK_AGENT_ADDRESS` environment variables to specify which address the agent listens on. The `--port` or `STORK_AGENT_PORT` environment variables specify which TCP port the agent listens on.

Normally, the agent will create a TCP socket on which to listen for commmands from a stork-server and create exporters which export data to Prometheus. There are two command line flags which may be used to alter this behavior. The `--listen-stork-only` flag instructs the agent to listen for commands from the Stork Server but not for Prometheus requests. Conversely, the `--listen-prometheus-only` flag instructs the agent to listen for Prometheus requests but not for commands from the Stork Server.

Note: Unless explicitly specified, the agent listens on all addresses on port 8080. There are no authentication mechanisms implemented in the agent yet. Use with care!

3.4 Connecting and Monitoring Machines

3.4.1 Registering a New Machine

Once the agent is deployed and running on the machine to be monitored, the `Stork Server` must be instructed to start monitoring it. This can be done via the `Services` menu, under `Machines`, to see a list of currently registered machines.

To add a new machine, click `Add New Machine` and specify the machine address (IP address, hostname, or FQDN) and a port.

After the `Add` button is clicked, the server attempts to establish a connection to the agent. Make sure that any active firewalls will allow incoming connections to the TCP port specified.

Once a machine is added, a number of parameters are displayed, including hostname, address, agent version, number of CPU cores, CPU load, available total memory, current memory utilization, uptime, OS, platform family, platform name, OS version, kernel, virtualization details (if any), and host ID.

If any applications, i.e. *Kea DHCP* and/or *BIND 9*, are detected on this machine, the status of those applications is displayed and the link allows navigation to the application details.

Navigation to the discovered applications is also possible through the `Services` menu.

3.4.2 Monitoring a Machine

Monitoring of registered machines is accomplished via the Services menu, under Machines. A list of currently registered machines is displayed, with multiple pages available if needed.

A filtering mechanism that acts as an omnibox is available. Via a typed string, Stork can search for an address, agent version, hostname, OS, platform, OS version, kernel version, kernel architecture, virtualization system, or host-id fields.

The state of a machine can be inspected by clicking its hostname; a new tab opens with the machine's details. Multiple tabs can be open at the same time, and clicking Refresh updates the available information.

The machine state can also be refreshed via the Action menu. On the Machines list, each machine has its own menu; click on the triple-lines button at the right side and choose the Refresh option.

3.4.3 Deleting a Machine

To stop monitoring a machine, go to the Machines list, find the machine to stop monitoring, click on the triple-lines button at the right side, and choose Delete. This will terminate the connection between the Stork server and the agent running on the machine, and the server will no longer monitor it. However, the Stork agent process will continue running on the machine. Complete shutdown of a Stork agent process must be done manually, e.g. by connecting to the machine using ssh and stopping the agent there. One way to achieve that is to issue the `killall stork-agent` command.

3.5 Monitoring Applications

3.5.1 Application Status

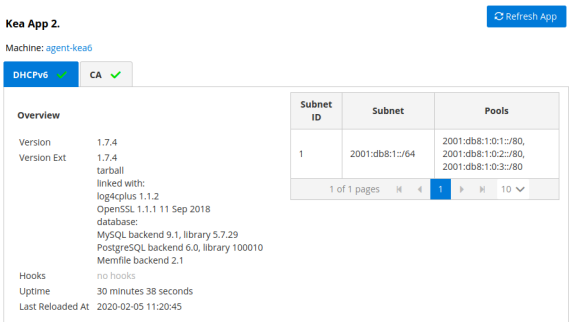
Kea DHCP and BIND 9 applications discovered on connected machines are listed via the top-level menu bar, under `Services`. Both the Kea and BIND 9 applications can be selected; the list view includes the application version, application status, and some machine details. The `Action` button is also available, to refresh the information about the application.

The application status displays a list of daemons belonging to the application. For BIND 9, it is always only one daemon, `named`. In the case of Kea, several daemons may be presented in the application status column, typically: DHCPv4, DHCPv6, DDNS, and CA (Kea Control Agent). The listed daemons are those that Stork finds in the CA configuration file. A warning sign is displayed for any daemons from the CA configuration file that are not running. In cases when the Kea installation is simply using the default CA configuration file, which includes configuration of daemons that are never intended to be launched, it is recommended to remove (or comment out) those configurations to eliminate unwanted warnings from Stork about inactive daemons.

3.5.2 IPv4 and IPv6 Subnets per Kea Application

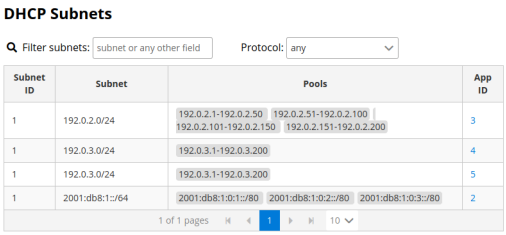
One of the primary configuration aspects of any network is the layout of IP addressing. This is represented in Kea with IPv4 and IPv6 subnets. Each subnet represents addresses used on a physical link. Typically, certain parts of each subnet ("pools") are delegated to the DHCP server to manage. Stork is able to display this information.

One way to inspect the subnets and pools within Kea is by looking at each Kea application to get an overview of what configurations a specific Kea application is serving. A list of configured subnets on that specific Kea application is displayed. The following picture shows a simple view of the Kea DHCPv6 server running with a single subnet, with three pools configured in it.



3.5.3 IPv4 and IPv6 Subnets in the Whole Network

It is convenient to see the complete overview of all subnets configured in the network being monitored by Stork. Once at least one machine with the Kea application running is added to Stork, click on the DHCP menu and choose Subnets to see all available subnets. The view shows all IPv4 and IPv6 subnets with the address pools and links to the applications that are providing them. An example view of all subnets in the network is presented in the figure below.



There are filtering capabilities available in Stork; it is possible to choose whether to see IPv4 only, IPv6 only, or both. There is also an omnisearch box available where users can type a search string. Note that for strings of four characters or more, the filtering takes place automatically, while shorter strings require the user to hit Enter. For example, in the above situation it is possible to show only the first (192.0.2.0/24) subnet by searching for the *0.2* string. One can also search for specific pools, and easily filter the subnet with a specific pool, by searching for part of the pool ranges, e.g. *3.200*.

Stork is able to display pool utilization for each subnet, and displays the absolute number of addresses allocated and percentage of usage. There are two thresholds: 80% (warning; the pool utilization bar becomes orange) and 90% (critical; the pool utilization bar becomes red).

Note: As of Stork 0.5.0, if two or more servers are handling the same subnet (e.g. a HA pair), the same subnet is listed multiple times. This limitation will be addressed in future releases.

3.5.4 IPv4 and IPv6 Networks

Kea uses the concept of a shared network, which is essentially a stack of subnets deployed on the same physical link. Stork is able to retrieve information about shared networks and aggregate it across all configured Kea servers. The Shared Networks view allows for the inspection of networks and the subnets that belong in them. Pool utilization is shown for each subnet.

3.5.5 Host Reservations

Kea DHCP servers can be configured to assign static resources or parameters to the DHCP clients communicating with the servers. Most commonly these resources are the IP addresses or delegated prefixes. However, Kea also allows for assigning hostnames, PXE boot parameters, client classes, DHCP options, and others. The mechanism by which a given set of resources and/or parameters is associated with a given DHCP client is called "host reservations."

A host reservation consists of one or more DHCP identifers used to associate the reservation with a client, e.g. MAC address, DUID, or client identifier; and a collection of resources and/or parameters to be returned to the client if the client's DHCP message is associated with the host reservation by one of the identifiers. Stork can detect existing host reservations specified both in the configuration files of the monitored Kea servers and in the host database backends accessed via the Kea host_cmds premium hooks library. At present, Stork provides no means to update or delete host reservations.

All reservations detected by Stork can be listed by selecting the `DHCP` menu option and then selecting `Hosts`.

The first column in the presented view displays one or more DHCP identifiers for each host in the format `hw-address=0a:1b:bd:43:5f:99`, where `hw-address` is the identifier type. In this case, the identifier type is the MAC address of the DHCP client for which the reservation has been specified. Supported identifier types are described in the following sections of the Kea ARM: Host Reservation in DHCPv4 and Host Reservation in DHCPv6. If multiple identifiers are present for a reservation, the reservation will be assigned when at least one of the identifiers matches the received DHCP packet.

The second column, `IP Reservations`, includes the static assignments of the IP addresses and/or delegated prefixes to the clients. There may be one or more IP reservations for each host.

The `Hostname` column contains an optional hostname reservation, i.e. the hostname assigned to the particular client by the DHCP servers via the Hostname or Client FQDN option.

The `Global/Subnet` column contains the prefixes of the subnets to which the reserved IP addresses and prefixes belong. If the reservation is global, i.e. is valid for all configured subnets of the given server, the word "global" is shown instead of the subnet prefix.

Finally, the `AppID @ Machine` column includes one or more links to Kea applications configured to assign each reservation to the client. The number of applications will typically be greater than one when Kea servers operate in the High Availability setup. In this case, each of the HA peers uses the same configuration and may allocate IP addresses and delegated prefixes to the same set of clients, including static assignments via host reservations. If HA peers are configured correctly, the reservations they share will have two links in `AppID @ Machine` column. Next to each link there is a little label indicating whether the host reservation for the given server has been specified in its configuration file or a host database (via host_cmds premium hooks library).

The `Filter hosts` input box is located above the Hosts table. It allows for filtering the hosts by identifier types, identifier values, IP reservations, hostnames and by globality i.e. `is:global` and `not:global`. When filtering by DHCP identifier values, it is not necessary to use colons between the pairs of hexadecimal digits. For example, the reservation `hw-address=0a:1b:bd:43:5f:99` will be found regardless of whether the filtering text is `1b:bd:43` or `1bbd43`.

3.5.6 Sources of Host Reservations

There are two ways to configure the Kea servers to use host reservations. First, the host reservations can be specified within the Kea configuration files; see Host Reservation in DHCPv4 for details. The other way is to use a host database backend, as described in Storing Host Reservations in MySQL, PostgreSQL, or Cassandra. The second solution requires the given Kea server to be configured to use the host_cmds premium hooks library. This library implements control commands used to store and fetch the host reservations from the host database which the Kea server is connected to. If the host_cmds hooks library is not loaded, Stork will only present the reservations specified within the Kea configuration files.

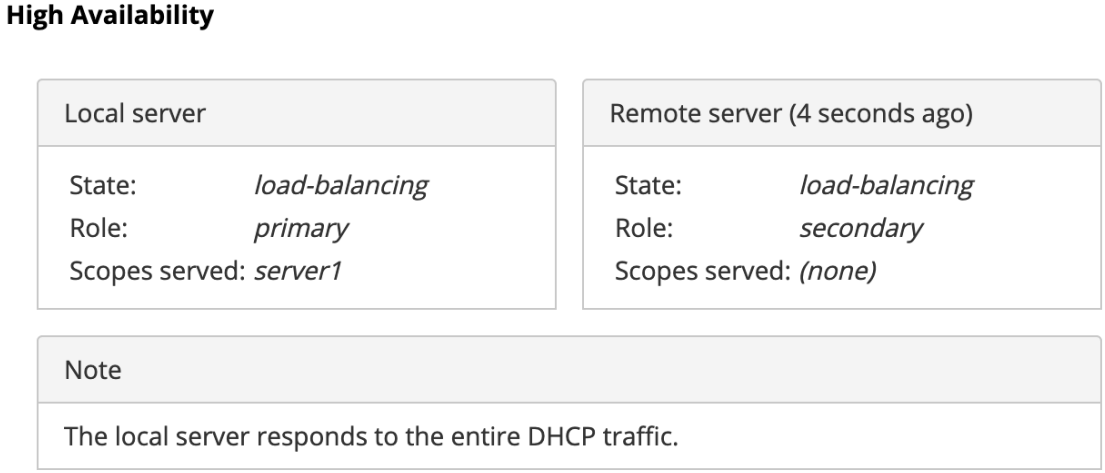
Stork periodically fetches the reservations from the host database backends and updates them in the local database. The default interval at which Stork refreshes host reservation information is set to 60 seconds. This means that an update in the host reservation database will not be visible in Stork until up to 60 seconds after it was applied. This interval is currently not configurable.

Note: As of the Stork 0.7.0 release, the list of host reservations must be manually refreshed by reloading the browser page to observe the most recent updates fetched from the Kea servers.

3.5.7 Kea High Availability Status

When viewing the details of the Kea application for which High Availability is enabled (via the libdhcp_ha.so hooks library), the High Availability live status is presented and periodically refreshed for the DHCPv4 and/or DHCPv6 daemon configured as primary or secondary/standby server. The status is not displayed for the server configured as an HA backup. See the High Availability section in the Kea ARM for details about the roles of the servers within the HA setup.

The following picture shows a typical High Availability status view displayed in the Stork UI.



The local server is the DHCP server (daemon) belonging to the application for which the status is displayed; the remote server is its active HA partner. The remote server belongs to a different application running on a different machine, and this machine may or may not be monitored by Stork. The statuses of both the local and the remote server are fetched by sending the status-get command to the Kea server whose details are displayed (the local server). In the load-balancing and hot-standby modes the local server periodically checks the status of its partner by sending the

`ha-heartbeat` command to it. Therefore, this information is not always up-to-date; its age depends on the heartbeat command interval (typically 10 seconds). The status of the remote server includes the age of the data displayed.

The status information contains the role, state, and scopes served by each HA partner. In the usual HA case, both servers are in load-balancing state, which means that both are serving DHCP clients and there is no failure. If the remote server crashes, the local server transitions to the partner-down state, which will be reflected in this view. If the local server crashes, this will manifest itself as a communication problem between Stork and the server.

As of Stork 0.8.0 release, the High Availability view may also contain the information about the heartbeat status between the two servers and the information about the failover progress. This information is only available while monitoring Kea 1.7.8 versions and later.

The failover progress information is only presented when one of the active servers has been unable to communicate with the partner via the heartbeat exchange for a time exceeding the max-heartbeat-delay threshold. If the server is configured to monitor the DHCP traffic directed to the partner to verify that the partner is not responding to this traffic before transitioning to the partner-down state, the information about the number of unacked clients (clients which failed to get the lease), connecting clients (all clients currently trying to get the lease from the partner) and the number of analyzed packets are displayed. The system administrator may use this information to diagnose why the failover transition has not taken place or when such transition is likely to happen.

More about High Availability status information provided by Kea can be found in the Kea ARM.

3.6 Dashboard

The Main Stork page presents a dashboard. It contains a panel with information about DHCP and a panel with events observed or noticed by Stork server.

3.6.1 DHCP Panel

DHCP panel includes two sections: one for DHCPv4 and one for DHCPv6. Each section contains 3 kinds of information:

- list of up to 5 subnets with the highest pool utilization
- list of up to 5 shared networks with the highest pool utilization
- statistics about DHCP

3.6.2 Events Panel

Events panel presents the list of the most recent events captured by the Stork server. There are 3 severity levels of the events: info, warning and error. Events pertaining to the particular entities, e.g. machines or applications, provide a link to a web page containing the information about the given object.

CHAPTER 4

Backend API

Stork Agent provides a REST API. The API is generated using [Swagger](https://swagger.io/). The API points are currently documented in the `api/swagger.yaml` file.

Note: In future Stork releases, the API documentation will be generated automatically.

CHAPTER 5

Developer's Guide

Note: We acknowledge that users and developers have different needs, so the user and developer documents should eventually be separated. However, since the project is still in its early stages, this section is kept in the Stork ARM for convenience.

5.1 Rakefile

Rakefile is a script for performing many development tasks like building source code, running linters, running unit tests, and running Stork services directly or in Docker containers.

There are several other Rake targets. For a complete list of available tasks, use *rake -T*. Also see the Stork wiki for detailed instructions.

5.2 Generating Documentation

To generate documentation, simply type `rake doc`. Sphinx and rtd-theme must be installed. The generated documentation will be available in the `doc/singlehtml` directory.

5.3 Setting Up the Development Environment

The following steps install Stork and its dependencies natively, i.e. on the host machine, rather than using Docker images.

First, PostgreSQL must be installed. This is OS-specific, so please follow the instructions from the *Installation* chapter.

Optional step: to initialize the database directly, the migrations tool must be built and used to initialize and upgrade the database to the latest schema. However, this is completely optional, as the database migration is triggered automatically

upon server startup. This is only useful if for some reason it is desirable to set up the database but not yet run the server. In most cases this step can be skipped.

```
$ rake build_migrations
$ backend/cmd/stork-db-migrate/stork-db-migrate init
$ backend/cmd/stork-db-migrate/stork-db-migrate up
```

Once the database environment is set up, the next step is to build all the tools. Note the first command below downloads some missing dependencies and installs them in a local directory. This is done only once and is not needed for future rebuilds, although it is safe to rerun the command.

```
$ rake build_backend
$ rake build_ui
```

The environment should be ready to run! Open three consoles and run the following three commands, one in each console:

```
$ rake run_server
```

```
$ rake serve_ui
```

```
$ rake run_agent
```

Once all three processes are running, connect to http://localhost:8080 via a web browser. See *Using Stork* for initial password information or for adding new machines to the server.

The *run_agent* runs the agent directly on the current operating system, natively; the exposed port of the agent is 8888.

There are other Rake tasks for running preconfigured agents in Docker containers. They are exposed to the host on specific ports.

When these agents are added as machines in the `Stork Server` UI, both a localhost address and a port specific to a given container must be specified. This is a list of ports for particular Rake tasks and containers:

- *rake run_kea_container*: Kea with DHCPv4, port 8888
- *rake run_kea6_container*: Kea with DHCPv6, port 8886
- *rake run_kea_ha_containers* (2 containers): Kea 1 and 2 with preconfigured HA, ports 8881 and 8882
- *rake run_bind9_container*: port 9999

5.3.1 Installing Git Hooks

There is a simple git hook that inserts the issue number in the commit message automatically; to use it, go to the `utils` directory and run the `git-hooks-install` script. It will copy the necessary file to the `.git/hooks` directory.

5.4 Agent API

The connection between the server and the agents is established using gRPC over http/2. The agent API definition is kept in the `backend/api/agent.proto` file. For debugging purposes, it is possible to connect to the agent using the grpcurl tool. For example, a list of currently provided gRPC calls may be retrieved with this command:

```
$ grpcurl -plaintext -proto backend/api/agent.proto localhost:8888 describe
agentapi.Agent is a service:
service Agent {
  rpc detectServices ( .agentapi.DetectServicesReq ) returns ( .agentapi.
↪DetectServicesRsp );
  rpc getState ( .agentapi.GetStateReq ) returns ( .agentapi.GetStateRsp );
  rpc restartKea ( .agentapi.RestartKeaReq ) returns ( .agentapi.RestartKeaRsp );
}
```

Specific gRPC calls can also be made. For example, to get the machine state, the following command can be used:

```
$ grpcurl -plaintext -proto backend/api/agent.proto localhost:8888 agentapi.Agent.
↪getState
{
  "agentVersion": "0.1.0",
  "hostname": "copernicus",
  "cpus": "8",
  "cpusLoad": "1.68 1.46 1.28",
  "memory": "16",
  "usedMemory": "59",
  "uptime": "2",
  "os": "darwin",
  "platform": "darwin",
  "platformFamily": "Standalone Workstation",
  "platformVersion": "10.14.6",
  "kernelVersion": "18.7.0",
  "kernelArch": "x86_64",
  "hostID": "c41337a1-0ec3-3896-a954-a1f85e849d53"
}
```

5.5 ReST API

The primary user of the ReST API is the Stork UI in a web browser. The definition of the ReST API is located in the api folder and is described in Swagger 2.0 format.

The description in Swagger is split into multiple files. Two files comprise a tag group:

- *-paths.yaml - defines URLs
- *-defs.yaml - contains entity definitions

All these files are combined by the `yamlinc` tool into a single Swagger file `swagger.yaml`. Then, `swagger.yaml` generates code for:

- the UI fronted by swagger-codegen
- the backend in Go lang by go-swagger

All these steps are accomplished by Rakefile.

5.6 Backend Unit Tests

There are unit tests for backend part (agent and server) written in Go. They can be run using Rake:

```
$ rake unittest_backend
```

This requires preparing a database in PostgreSQL. One way to avoid doing this manually is by using a docker container with PostgreSQL which is automatically created when running the following Rake task:

```
$ rake unittest_backend_db
```

This one task spawns a container with PostgreSQL in the background and then it runs unit tests. When the tests are completed the database is shutdown and removed.

5.6.1 Unit Tests Database

When docker container with a database is not used for unit tests, the PostgreSQL server must be stared and the following role must be created:

```
postgres=# CREATE USER storktest WITH PASSWORD 'storktest';
CREATE ROLE
postgres=# ALTER ROLE storktest SUPERUSER;
ALTER ROLE
```

To point unit tests to our specific database set `POSTGRES_ADDR` environment variable, e.g.:

```
$ rake unittest_backend POSTGRES_ADDR=host:port
```

By default it points to `localhost:5432`.

5.6.2 Unit Tests Coverage

At the end of tests execution there is coverage report presented. If coverage of any module is below a threshold of 35% then an error is raised.

5.7 Docker Containers

To ease testing, there are several Docker containers available.

- `server` - This container is essential. It runs the Stork server, which interacts with all the agents and the database and exposes the API. Without it, Stork will not be able to function.
- `postgres` - This container is essential. It runs the PostgreSQL database that is used by the Stork server. Without it, the Stork server will produce error messages about an unavailable database.
- `webui` - This container is essential in most circumstances. It provides the front-end web interface. It is potentially unnecessary with the custom development of a Stork API client.

There are also several containers provided that are used to samples and they are not strictly necessary. The following containers will not be needed in a production network, however they're very useful to demonstrate existing Stork capabilities. They simulate certain services that Stork is able to handle:

- `agent-bind9` - This container runs a BIND 9 server. With this container, the agent can be added as a machine and Stork will begin monitoring its BIND 9 service.
- `agent-bind9-2` - This container also runs a BIND 9 server, for the purpose of experimenting with two different DNS servers.
- `agent-kea` - This container runs a Kea DHCPv4 server. With this container, the agent can be added as a machine and Stork will begin monitoring its Kea DHCPv4 service.

- `agent-kea-ha1` and `agent-kea-ha2` - These two containers should, in general, be run together. They each have a Kea DHCPv4 server instance configured in a HA pair. With both running and registered as machines in Stork, users can observe certain HA mechanisms, such as one taking over the traffic if the partner becomes unavailable.
- `traffic-dhcp` - This container is optional. If started, it can be used to transmit DHCP packets to `agent-kea`. It may be useful to observe non-zero statistics coming from Kea. When running Stork in Docker, `rake start_traffic_dhcp` can be used to conveniently control traffic.
- `traffic-dns` - This container is optional. If stated, it can be used to transmit DNS packets towards agent-bind9. It may be useful to observe non-zero statistics coming from BIND 9. If you're running Stork in docker, you can conveniently control that using `rake start_traffic_dns`.
- `prometheus` - This is a container with Prometheus for monitoring applications. It is preconfigured to monitor Kea and BIND 9 containers.
- `grafana` - This is a container with Grafana, a dashboard for Prometheus. It is preconfigured to pull data from a Prometheus container and show Stork dashboards.

5.8 Packaging

There are scripts for packaging the binary form of Stork. There are two supported formats:

- RPM
- deb

The RPM package is built on the latest CentOS version. The deb package is built on the latest Ubuntu LTS.

There are two packages built for each system: a server and an agent.

There are Rake tasks that perform the entire build procedure in a Docker container: *build_rpms_in_docker* and *build_debs_in_docker*. It is also possible to build packages directly in the current operating system; this is provided by the *deb_agent*, *rpm_agent*, *deb_server*, and *rpm_server* Rake tasks.

Internally, these packages are built by FPM (https://fpm.readthedocs.io/). The containers that are used to build packages are prebuilt with all dependencies required, using the *build_fpm_containers* Rake task. The definitions of these containers are placed in *docker/pkgs/centos-8.txt* and *docker/pkgs/ubuntu-18-04.txt*.

CHAPTER 6

Demo

A demo installation of `Stork` can be used to demonstrate `Stork` capabilities but can be used for its development as well.

The demo installation uses *Docker* and *Docker Compose* to set up all *Stork* services. It contains:

- Stork Server
- Stork Agent with Kea DHCPv4
- Stork Agent with Kea DHCPv6
- Stork Agent with Kea HA-1 (high availability server 1)
- Stork Agent with Kea HA-2 (high availability server 2)
- Stork Agent with BIND 9
- Stork DHCP Traffic Simulator
- Stork DNS Traffic Simulator
- PostgreSQL database
- Prometheus & Grafana

These services allow observation of many Stork features.

6.1 Requirements

Running the `Stork Demo` requires the same dependencies as building Stork, which is described in the *Installing from Sources* chapter.

Besides the standard dependencies, the `Stork Demo` requires:

- Docker
- Docker Compose

For details, please see the Stork wiki https://gitlab.isc.org/isc-projects/stork/wikis/Development-Environment.

6.2 Installation Steps

The following command retrieves all required software (go, goswagger, nodejs, Angular dependencies, etc.) to the local directory. No root password is necessary. Then it prepares Docker images and starts them up.

```
$ rake docker_up
```

Once the build process finishes, the Stork UI is available at http://localhost:8080/. Use any browser to connect.

The installation procedure creates several Docker images:

- *stork_webui*: a web UI interface,
- *stork_server*: a server backend,
- *postgres*: a PostgreSQL database used by the server,
- *stork_agent-bind9*: an agent with BIND 9,
- *stork_agent-bind9-2*: a second agent with BIND 9,
- *stork_agent-kea*: an agent with a Kea DHCPv4 server,
- *stork_agent-kea6*: an agent with a Kea DHCPv6 server,
- *stork_agent-kea-ha1*: the primary Kea DHCPv4 server in High Availability mode,
- *stork_agent-kea-ha2*: the secondary Kea DHCPv4 server in High Availability mode,
- *traffic-dhcp*: a web application that can run DHCP traffic using perfdhcp,
- *traffic-dns*: a web application that can run DNS traffic using dig and flamethrower,
- *prometheus*: Prometheus, a monitoring solution (https://prometheus.io/),
- *grafana*: Grafana, a dashboard for Prometheus (https://grafana.com/)

Note: The containers running the Kea and BIND 9 applications are for demo purposes only. They allow users to quickly start experimenting with Stork without having to manually deploy Kea and/or BIND 9 instances.

The PostgreSQL database schema is automatically migrated to the latest version required by the Stork server process.

The installation procedure assumes those images are fully under Stork control. If there are existing images, they will be overwritten.

6.2.1 Premium Features

It is possible to run the demo with premium features enabled in Kea apps. It requires starting the demo with an access token to the Kea premium repositories. Access tokens can be found on https://cloudsmith.io/~isc/repos/kea-1-7-prv/setup/#formats-deb. The token can be found inside this URL on that page: `https://dl.cloudsmith.io/<access token>/isc/kea-1-7-prv/cfg/setup/bash.deb.sh`. This web page and the token are available only to ISC employees and paid customers of ISC.

```
$ rake docker_up cs_repo_access_token=<access token>
```

6.3 Initialization

`Stork Server` requires some initial information:

1. Go to http://localhost:8080/machines/all
2. Add new machines (leave the default port):
    1. agent-kea
    2. agent-kea6
    3. agent-kea-ha1
    4. agent-kea-ha2
    5. agent-bind9
    6. agent-bind9-2

6.4 DHCP Traffic Simulator

The traffic simulator allows DHCP traffic to be sent to selected subnets pre-configured in Kea instances, with a limitation: it is possible to send traffic to only one subnet from a given shared network.

The traffic simulator can be found at: http://localhost:5000/

6.5 DNS Traffic Simulator

Traffic simulator allows sending DNS traffic to selected DNS servers.

Traffic simulator can be found at: http://localhost:5001/

6.6 Prometheus

The Prometheus instance is preconfigured and pulls statistics from:

- node exporters: agent-kea:9100, agent-bind9:9100, agent-bind9:9100
- kea exporters embedded in stork-agent: agent-kea:9547, agent-kea6:9547, agent-kea-ha1:9547, agent-kea-ha2:9547
- bind exporters embedded in stork-agent: agent-bind9:9119, agent-bind9-2:9119

The Prometheus web page can be found at: http://localhost:9090/

6.7 Grafana

The Grafana instance is preconfigured as well. It pulls data from Prometheus and loads dashboards from the Stork repository, in the Grafana folder.

The Grafana web page can be found at: http://localhost:3000/

CHAPTER 7

Manual Pages

7.1 stork-server - The central Stork server

7.1.1 Synopsis

stork-server

7.1.2 Description

The `stork-server` provides the main Stork server capabilities. In every stork deployment, there should be exactly one stork-server.

7.1.3 Arguments

The Stork Server takes the following arguments:

-h or --help Displays list of available parameters.

-v or --version Returns stork-agent version.

-d or --db-name= The name of the database to connect to (default: stork) [$STORK_DATABASE_NAME]

-u or --db-user The user name to be used for database connections (default: stork) [$STORK_DATABASE_USER_NAME]

--db-host The name of the host where database is available (default: localhost) [$STORK_DATABASE_HOST]

-p or --db-port The port on which the database is available (default: 5432) [$STORK_DATABASE_PORT]

--db-trace-queries Enable tracing SQL queries [$STORK_DATABASE_TRACE]

--rest-cleanup-timeout grace period for which to wait before killing idle connections (default: 10s)

--rest-graceful-timeout grace period for which to wait before shutting down the server (default: 15s)

--rest-max-header-size controls the maximum number of bytes the server will read parsing the request header's keys and values, including the request line. It does not limit the size of the request body. (default: 1MiB)

--rest-host the IP to listen on [$STORK_REST_HOST]

--rest-port the port to listen on for connections (default: 8080) [$STORK_REST_PORT]

--rest-listen-limit limit the number of outstanding requests

--rest-keep-alive set the TCP keep-alive timeouts on accepted connections. It prunes dead TCP connections (e.g. closing laptop mid-download) (default: 3m)

--rest-read-timeout maximum duration before timing out read of the request (default: 30s)

--rest-write-timeout maximum duration before timing out write of the response (default: 60s)

--rest-tls-certificate the certificate to use for secure connections [$STORK_REST_TLS_CERTIFICATE]

--rest-tls-key the private key to use for secure connections [$STORK_REST_TLS_PRIVATE_KEY]

--rest-tls-ca the certificate authority file to be used with mutual tls auth [$STORK_REST_TLS_CA_CERTIFICATE]

--rest-static-files-dir Directory with static files for UI [$STORK_REST_STATIC_FILES_DIR]

7.1.4 Mailing List and Support

There is a public mailing list available for the Stork project. **stork-dev** (stork-dev at lists.isc.org) is intended for Kea developers, prospective contributors, and other advanced users. The list is available at https://lists.isc.org. The community provides best-effort support on both of those lists.

Once stork will become more mature, ISC will be providing professional support for Stork services.

7.1.5 History

The `stork-server` was first coded in November 2019 by Michal Nowikowski and Marcin Siodelski.

7.1.6 See Also

stork-agent(8)

7.2 stork-agent - Stork agent that monitors BIND 9 and Kea services

7.2.1 Synopsis

stork-agent [**–host**] [**–port**]

7.2.2 Description

The `stork-agent` is a small tool that is being run on the systems that are running BIND 9 and Kea services. Stork server connects to the stork agent and uses it to monitor services remotely.

7.2.3 Arguments

The Stork Agent takes the following arguments:

-h or --help Displays list of available parameters.

-v or --version Returns stork-agent version.

--host=hostname Specifies the IP to listen on. Can be controlled with $STORK_AGENT_ADDRESS environment variable. The default value is `::`.

--port=1234 Specifies the TCP port to listen on for connections. The default is 8080. Can be controlled with $STORK_AGENT_PORT environment variable.

--listen-stork-only Instructs the agent to listen for commands from the Stork Server but not for Prometheus requests. Can also be set with the $STORK_AGENT_LISTEN_STORK_ONLY environment variable.

--listen-prometheus-only Instructs the agent to listen for Prometheus requests but not for commands from the Stork Server. Can also be set with the $STORK_AGENT_LISTEN_PROMETHEUS_ONLY environment variable.

--prometheus-kea-exporter-host Instructs the agent to open Prometheus Kea exporter socket on specified address.

--prometheus-kea-exporter-port Instructs the agent to open Prometheus Kea exporter socket on specified port.

--prometheus-kea-exporter-interval Instruct the agent to how frequently the statistics should be pulled from Kea.

--prometheus-bind9-exporter-host Instructs the agent to open Prometheus BIND 9 exporter socket on specified address.

--prometheus-bind9-exporter-port Instructs the agent to open Prometheus BIND 9 exporter socket on specified port.

--prometheus-bind9-exporter-interval Instruct the agent to how frequently the statistics should be pulled from BIND 9.

7.2.4 Configuration

Stork agent uses two environment variables to control its behavior:

- STORK_AGENT_ADDRESS - if defined, governs which IP address to listen on
- STORK_AGENT_PORT - if defined, it controls which port to listen on. The default is 8080.

7.2.5 Mailing List and Support

There is a public mailing list available for the Stork project. **stork-dev** (stork-dev at lists.isc.org) is intended for BIND 9 and Kea developers, prospective contributors, and other advanced users. The list is available at https://lists.isc.org. The community provides best-effort support on both of those lists.

Once stork will become more mature, ISC will be providing professional support for Stork services.

7.2.6 History

The `stork-agent` was first coded in November 2019 by Michal Nowikowski.

7.2.7 See Also

stork-server(8)

CHAPTER 8

Indices and tables

- genindex
- modindex
- search