Stork

Release 0.5.0

Mar 08, 2020

Contents

Stork is a new project proposed by ISC with the aim of delivering BIND 9 and Kea dashboard. It is going to be a spiritual successor of earlier attempts - Kittiwake and Antherius. It is currently in very early stages of planning.

This is the reference guide for Stork version 0.5.0. Links to the most up-to-date version of this document, along with other documents for Stork, can be found in ISC's Stork project homepage or readthedocs .

CHAPTER 1

Stork Installation

Stork is in its very early stages of development. As such, it is currently only supported on Ubuntu 18.04. It is likely that the code would work on many other systems, but for the time being we want to focus on the core development, rather than portability issues.

There are several dependencies that needs to be installed:

- rake
- Java Runtime Environment
- Docker and Docker Compose (if installing using Docker)

For details, please see Stork wiki https://gitlab.isc.org/isc-projects/stork/wikis/Development-Environment . Note the Stork project is in very early stages and its building instructions change frequently. Please refer to the wiki page in case of problems.

Docker is currently the most convenient way to run Stork easily. However, it possible to compile and run Stork without it. For details, see the next section.

Java is currently a build-time dependency, because one of the tools used to generate API bindings, swagger-codegen, is written in Java. However, Java is not needed to run Stork. In the future Stork versions, this dependency will be optional, only necessary for developers who want to implement new or change existing API interfaces.

For ease of deployment, Stork uses Rake to automate compilation and installation. It facilitates installation both using Docker and without Docker (see the following sections).

1.1 Installation Steps

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

First, you need to install PostgreSQL. This is OS specific. Please follow up the instructions for your system.

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

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

Optional step: if you want to initialize the database on your own, you need to build the migrations and use it to initialize and upgrade the DB to the latest schema. However, this is completely optional as the database migration will be triggered automatically upon the server startup. This is only useful if for some reason you want to set up the database, but don't want to run the server yet. 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
```

Now that you have the database environment set up, the next step is to build all the tools. Note the first command will download some missing dependencies needed and will install it in a local directory. This is done only once and is not needed for future rebuilds. However, it's safe to rerun the command.

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

The environment should be ready to run! Open 3 consoles, and run the following 3 commands, one in each console:

```
$ rake run_server
$ rake serve_ui
$ rake run_agent
```

Once all three processes are running, go ahead and connect to http://localhost:4200 with your web browser. See *Using Stork* for initial password information.

CHAPTER 2

Using Stork

This section describes how to use features available in stork. To connect to Stork, use your web browser and connect to port 4200. If Stork is running on your localhost, you can navigate to http://localhost:4200.

2.1 Managing users

Upon the initial installation the default administrator's account is created and can be used to sign in to the system via the web UI. Please use the login `admin` and password `admin` to sign in to the system.

To manage users, click on the `Configuration` menu and choose `Users`. You will see a list of existing users. At the very least, there will be user `admin`.

To add new user, click `Create User Account`. A new tab will opened that will let you specify the new account parameters. Some fields have specific restrictions. Username can consist of only letters, numbers and underscore. E-mail field is optional. However, if specified, it must be a well formed e-mail. First and lastname fields are mandatory. Password must only contain letters, digits, @, ., !, +, - and must be at least 8 characters long.

Currently, the users are be 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. The 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 belonging to 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 will become active and you will be able to add new account.

2.2 Changing User Password

Initial password is assigned by the administrator when the user account is created. Each user should change the password when he or she first logs in to the system. Click on the `Profile` menu and choose `Settings`. The user profile information is displayed. Click on `Change password` in the menu bar on the left. In the first input box the current password must be specified. The new password must be specified in the second input box and this password must meet the normal requirements for the password as mentioned in the previous sections. Finally, the password must

be confirmed in the third input box. When all entered data is valid the `Save` button will be activated. Clicking this button will attempt to change the password.

2.3 Deploying Stork Agent

Stork system uses agents to monitor services. Stork Agent (*STAG* or simply *agent*) is a daemon that is expected to be deployed and run on each machine to be monitored. Currently, there are no automated deployment routines and STAG has to be copied and run manually. This can be done in a variety of ways. Here is one of them.

Assuming you want to monitor services running on machine with IP 192.0.2.1, you can do 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, you need to start the agent. In the basic case, you can simply run it:

```
./stork-agent
```

You can optionally pass `--host=` or set the `STORK_AGENT_ADDRESS` environment variable to specify which address the agent will listen on. You can pass `--port` or set the `STORK_AGENT_PORT` environment variable to specify which TCP port the agent will listen on.

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

2.4 Connecting and Monitoring Machines

2.4.1 Registering New Machine

Once the agent is deployed and running on the machine to be monitored, you should instruct Stork server to start monitoring it. You can do so by going to Services menu and choosing Machines. You will be presented with a list of currently registered machines.

To add a new machine, click `Add New Machine`. You need to specify the machine address or hostname and a port. If Stork agent is running in a container, you should specify the container name as a machine hostname. If you launched Stork using `rake docker_up` command you may specify one of the demo container names, e.g. agent-kea, agent-bind9 etc. The demo agents are running on port 8080. If the agent you're connecting to was launched using `rake run_agent` it will listen on localhost port 8888.

Once you click Add, the server will attempt to establish gRPC over http/2 connection to the agent. Make sure that any firewalls in between will allow incoming connections to the TCP port specified.

Once a machine is added, a number of parameters, such as 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), host ID and other information will be displayed.

If any applications, i.e. Kea or/and BIND 9 are detected on this machine, the status of those applications will be displayed and the link will allow for navigating to the application details.

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

2.4.2 Monitoring Machines

To monitor registered machines, go to Services menu and click Machines. A list of currently registered machines will be displayed. Pagination mechanism is available to display larger number of machines.

There is a filtering mechanism that acts as an omnibox. The string typed is searched for an address, agent version, hostname, OS, platform, OS version, kernel version, kernel architecture, virtualization system, host-id fields. The filtering happens once you hit ENTER.

You can inspect the state of a machine by clicking its hostname. A new tab will open with machine details. Multiple tabs can be open at the same time. You can click Refresh state to get updated information.

The machine state can also be refreshed using 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.

2.4.3 Deleting Machines

To stop monitoring a machine, you can go to the Machines list, find the machine you want to stop monitoring, click on the triple lines button at the right side and choose Delete. Note this will terminate the connection between 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. If you want to completely shut it down, you need to do so manually, e.g. by connecting to the machine using ssh and stopping the agent there. One way to achieve that is to issue `killall stork-agent` command.

2.5 Monitoring Applications

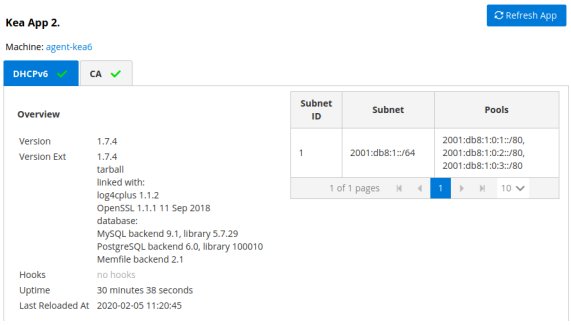
2.5.1 Application Status

Kea and BIND 9 applications discovered on the connected machines can be listed via the top level menu bar, under `Services`. You can select between Kea and BIND 9 applications. The list of applications of the given type comprises the application version, application status and some machine details. The `Action` button is also available which allows for refreshing the information about the application.

The application status comprises a list of daemons belonging to the application. For BIND 9 it is always only one daemon, `named`. In case of Kea, several daemons can be presented in the application status column, typically: DHCPv4, DHCPv6, DDNS and CA (Kea Control Agent). The listed daemons are those that Stork found in the CA configuration file. The warning sign will be displayed for those 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.

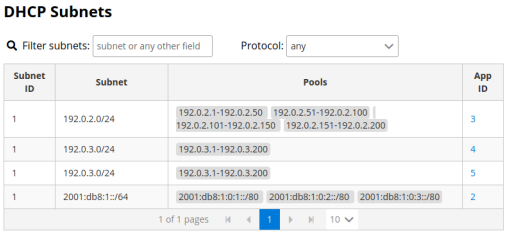
2.5.2 IPv4 and IPv6 Subnets per Kea Application

One of the primary configuration aspects of any network is how the IP addressing is laid out. This is represented in Kea with IPv4 and IPv6 subnets. Each subnet represents addresses being 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 of the ways to inspect the subnets and pools within is by looking at the Kea applications. This will give you an overview of what kind of configuration this specific Kea application is serving. A list of configured subnets on that specific Kea application will be displayed. The following picture shows a simple view of the Kea DHCPv6 server running with a single subnet with three pools configured in it.



2.5.3 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. To view all subnets, click on DHCP menu and choose Subnets. Note that you should have at least one machine added with Kea application running on it. The view shows all IPv4 and IPv6 subnets with the address pools and the links to applications that are providing them. An example view of all subnets in the network is presented in figure below.



There are filtering capabilities available. You can choose whether you want to see IPv4 only, IPv6 only or both. There is also omnisearch box available. You can type a string you are looking for. Note that for strings of 4 characters and more, the filtering takes place automatically. For shorter strings, you need to also hit Enter. For example, in the above situation you can choose to show only the first (192.0.2.0/24) subnet by searching for *0.2* string. You can also search for specific pools. For example, you can easily filter the subnet with specific pool if you search for part of the pool ranges, e.g. *3.200*.

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

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

2.5.4 IPv4 and IPv6 Networks

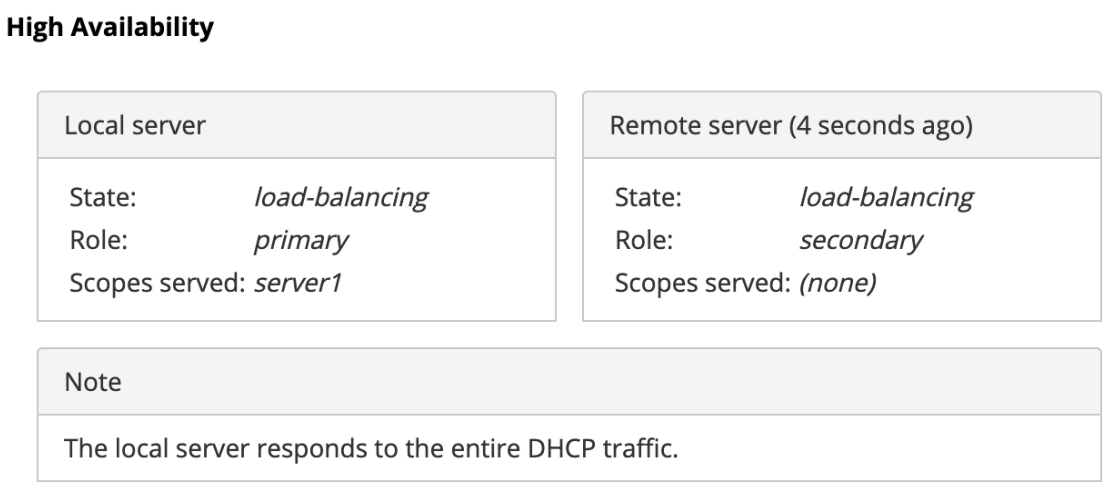
Kea has a concept of shared networks (or networks), which is essentially a stack of subnets deployed on the same physical link. This feature is very popular among users. Stork is now able to retrieve information about the shared networks and aggregate it across all configured Kea servers. The Shared Networks view allows for inspection of networks and the subnets that belong in them. Pool utilization is shown for each subnet.

2.5.5 Kea High Availability Status

When viewing the details of the Kea application for which High Availability is enabled (via libdhcp_ha.so hooks library), the High Availability live status is presented and periodically refreshed for the DHCPv4 and/or DHCPv6

deamon configured as primary or secondary/standby server. The status is not displayed for the server configured as a HA backup. See the High Availability section in the Kea ARM for the details about various roles of the servers within the HA setup.

The following picture shows a typical High Availability status view displayed in 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 server belongs to a different application running on a different machine and this machine may or may not be monitored by Stork. The status of both the local and the remote server is fetched by sending the status-get command to the Kea server which details are displayed (local server). 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 and its age depends on the heartbeat command interval (typically 10s). The status of the remote server includes the age of the data displayed.

The status information contains the role, state and the scopes served by each HA partner. In our case, both servers are in load-balancing state which means that both are serving the DHCP clients and there is no failure. If the remote server crashes, the local server should transition 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.

2.6 Dashboard

The Main Stork page presents a simple dashboard. It includes some statistics about the monitored applications such as: a total number of Kea and BIND 9 applications and a number of misbehaving applications.

CHAPTER 3

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

CHAPTER 4

Developer's Guide

Note: We acknowledge that users and developers are two different groups of people, so the documents should eventually be separated. However, since these are still very early days of the project, this section is kept in the Stork ARM for convenience only.

4.1 Rakefile

Rakefile is a script for performing many development tasks like building source code, running linters, running unit tests, 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 wiki for detailed instructions.

4.2 Generating Documentation

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

4.3 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 grpcurl tool. For example, you can retrieve a list of currently provided gRPC calls by using 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 );
}
```

You can also make specific gRPC calls. 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"
}
```

4.4 Installing git hooks

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

4.5 ReST API

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

The description in Swagger is split into multiple files. 2 files comprise a tag group:

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

All these files are combined by `yamlinc` tool into signle swagger file `swagger.yaml`. Then from `swagger.yaml` there are generated code for:

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

All these steps are realized by Rakefile.

4.6 Docker Containers

To ease testing, there are several docker containers available. Not all of them are necessary.

- `server` - This container is essential. It runs the Stork server, which interacts with all the agents, the database and exposes 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, Stork server will will only able to produce error messages about unavailable database.
- `webui` - This container is essential in most circumstances. It provides the front web interface. You could possibly not run it, if you are developing your own Stork API client.

There are also several containers provided that are used to samples. Those 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 BIND 9 server. If you run it, you can add it as a machine and Stork will begin monitoring its BIND 9 service.
- `agent-kea` - This container runs Kea DHCPv4 server. If you run it, you can add it as a machine and Stork will begin monitoring its BIND 9 service.
- `agent-kea-ha1` and `agent-kea-ha2` - Those two containers should in general be run together. They have each a Kea DHCPv4 server instance configured in a HA pair. If you run both of them and register them as machines in Stork, you will be able to observe certain HA mechanisms, such as one taking over the traffic if the partner becomes unavailable.
- `traffic-dhcp` - This container is optional. If stated, it will start transmitting DHCP packets towards agent-kea. It may be useful to observe non-zero statistics coming from Kea. If you're running Stork in docker, you can coveniently control that using `rake start_traffic_dhcp` and
- `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 Prometheus container and show Stork dashboards.

CHAPTER 5

Demo

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

Demo installation is using Docker Compose for setting 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
- PostgreSQL database
- Prometheus & Grafana

Running all these services allows presenting many features of Stork.

5.1 Installation steps

The following command will retrieve all required software (go, goswagger, nodejs, Angular dependencies, etc.) to your local directory. No root password necessary.

```
# Prepare docker images and start them up
rake docker_up
```

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

The installation procedure will create several Docker images:

- *stork_webui*: exposing web UI interface,

- *stork_server*: a server backend,
- *postgres*: a PostgreSQL database used by the server,
- *stork_agent-bind9*: agent with BIND 9,
- *stork_agent-kea*: agent with Kea DHCPv4 server,
- *stork_agent-kea6*: agent with 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
- *prometheus*: Prometheus, a monitoring solution (https://prometheus.io/)
- *grafana*: Grafana, a dashboard for Prometheus (https://grafana.com/)

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

The PostgreSQL database schema will be 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.

5.2 Requirements

Requirements are the same as for Stork development. For details, please see Stork wiki https://gitlab.isc.org/isc-projects/stork/wikis/Development-Environment .

5.3 Initialization

At the beginning some initial information needs to be added in Stork Server:

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

5.4 DHCP Traffic Simulator

Traffic simulator allows sending DHCP traffic to selected subnets pre-configured in Kea instances. There is a limitation, it is possible to send traffic to one subnet from given shared network.

Traffic simulator can be found at: http://localhost:5000/

5.5 Prometheus

Prometheus instance is preconfigured and pulls stats from:

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

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

5.6 Grafana

Grafana instance is preconfigured as well. It pulls data from Prometheus and loads dashboards from stork repository, from grafana folder.

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

CHAPTER 6

Manual Pages

6.1 stork-server - The central Stork server

6.1.1 Synopsis

stork-server

6.1.2 Description

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

6.1.3 Arguments

Currently stork-server takes no arguments.

6.1.4 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.

6.1.5 History

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

6.1.6 See Also

stork-agent(8)

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

6.2.1 Synopsis

stork-agent [**–host**] [**–port**]

6.2.2 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.

6.2.3 Arguments

The Stork Agent takes the following arguments:

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

--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.

6.2.4 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.

6.2.5 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.

6.2.6 History

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

6.2.7 See Also

stork-server(8)

CHAPTER 7

Indices and tables

- genindex
- modindex
- search