CheatSheet RabbitMQ v1.02

Erstellt am: 28. April 2020

Autor(en): Peter Dreuw
<peter.dreuw@credativ.de>

credativ GmbH
Trompeterallee 108
41189 Mönchengladbach
Tel. +49 2166 9901-0
www.credativ.de

Einführung

RabbitMQ ist eine von Pivotal Software gepflegte Open Source Software unter der Mozilla Public License. RabbitMQ wurde in Erlang geschrieben und wird als Message Orientated Middleware bzw. Message Broker eingesetzt. Es kann als Single-Server oder im Cluster mit verschiedenen Netzwerkstrukturen eingesetzt werden.

Das CLI: `rabbitmqctl`

Das CLI zu RabbitMQ unterstützt unter anderem folgende Parameter:

`--erlang-cookie`
: übergibt ein Erlangcookie anstelle der im Dateisystem abgelegten Cookiedatei (Siehe: Cluster).

`-n <name>@<knoten>`
: führt den Befehl nicht auf dem lokalen Knoten aus, sondern auf <name>@<knoten>. Hierzu muss für das CLI das Erlang-Cookie des Knotens gesetzt sein. Diese Option ist unabhängig von einem Clusterbetrieb des RabbitMQ und funktioniert auf Basis der Erlang-Runtime.

`--dry-run`
: führt den Befehl nicht aus, sondern gibt nur eine Information hierzu aus.

`-q | --quiet`
: unterdrückt Ausgaben bei Befehlsausführung.

Installation & Inbetriebnahme

RabbitMQ benötigt zum Betrieb eine Erlang / OTP-Installation. Diese muss den der jeweiligen Version benötigten Mindest-Versionsstand haben. In den meisten Fällen wird man die Erlang/OTP-Pakete sowie RabbitMQ mit den Paketen aus der Betriebssystemdistribution einsetzen. Vergl. `https://www.rabbitmq.com/which-erlang.html`

RabbitMQ Konfigdatei

Die `rabbitmq.conf` und `rabbitmq-env.conf` finden sich bei den meisten Linux-Distributionen unter `/etc/rabbitmq`, möglich ist auch `$RABBITMQ_HOME/etc/rabbitmq`. Unter Windows meist `%APPDATA%\rabbitmq`.
`rabbitmq.conf` ist ab Version 3.7 im sysctl-Format. Ältere Versionen beinhalten einen gültigen Erlangausdruck (Listen und Tupel). Ein kommentiertes Muster zum alten Konfigurationsformat findet sich unter `https://github.com/rabbitmq/rabbitmq-server/blob/v3.6.x/docs/rabbitmq.config.example`

PlugIns steuern

RabbitMQ durch Plugins erweitern:

`rabbitmq-plugins enable plugin`
: schaltet "plugin" ein

`rabbitmq-plugins disable plugin`
: schaltet "plugin" aus

`rabbitmq-plugins list`
: listet eingeschaltete PlugIns.

Hinweis: `rabbitmq-plugins` unterstützt den Parameter `-v` für "verbose" und `--offline`, der die Ausführung auf einem inaktiven Node erlaubt.

Nützliche PlugIns

`managment` administratives WebUI für RabbitMQ.

`shovel` verbindet zwei Messagebroker wie Consumer und Producer.

`shovel_management` WebUI zum Shovel-PlugIn.

`federation` verbindet Messagebroker ober WAN-Grenzen oder Versionsgrenzen.

`federation_management` WebUI zum Federation-PlugIn.

`lager` Konfigurierbares Logging (ab RabbitMQ Vers. 3.7 im Core integriert)

Alle "offiziellen" PlugIn-Namen beginnen mit "`rabbitmq_`". So wird z.B. aus `federation_management` aus der o.g. Liste `rabbitmq_federation_management`. Weitere offizielle PlugIns: LDAP, http-auth, SASL, AMPQ1.0, MQTT, STOMP, WebSTOMP, WebMQTT oder Queue-Sharding.

Netzwerkports

4369 Erlang peer discovery (nur Clusterbetrieb)

5672, 5671 AMQP Clients ohne und mit TLS

25672 Knoten-zu-Knoten- und CLI-zu-Knoten-Kommunikation

35672-35682 CLI-Tools

15672 WebUI des Clusters (nur: Management PlugIn enabled)

61613,61614 STOMP (nur: STOMP PlugIn enabled)

1883, 8883 MQTT ohne und mit TLS (nur: MQTT PlugIn enabled)

15674 WebSTOMP (nur mit PlugIns)

15675 WebMQTT (nur mit PlugIns)

Cluster

Verteilte RabbitMQ-Systeme

Über Federation- oder Shovel-Plugin sind verteilte Systeme möglich. Diese sind Unabhängig in Version, Benutzerdefinitionen/-rechten, tolerieren hohe Netzwerklatenzen, können verschiedene Topologien abbilden. Dieser Ansatz erhöht die Verfügbarkeit gegenüber der Konsistenz (CAP-Theorem).
Die Steuerung ist sehr individuel, daher hier der Verweis auf die Dokumentation zu Shovel- und Federation-Plugin.

Server-Cluster

Im RabbitMQ-Cluster muss jeder Knoten den gleichen Major/Minor-Versiosstand von RabbitMQ (und oft Erlang) haben. Für alle Knoten muss ein identisches Erlang-Cookie gesetzt sein. Alle Knoten des Clusters haben eine identische Benutzer- und Datenbasis, auch wenn nicht alle Knoten alle Daten lokal speichern müssen. Der Cluster stellt sich wie ein logischer Broker dar. Dieser Ansatz erhöht die Konsistenz gegenüber der Verfügbarkeit (CAP-Theorem).

Erlang-Cookie

Das Cookie ist eine simple Textdatei mit einem geheimen Inhalt in ASCII, etwa ein Hash etc. Dieser Inhalt ist das Shared Secret zwischen allen in das Cluster eingebundene Erlang Knoten.

- Unter Unix unter `/var/librabbitmq/.erlang-cookie` oder `$HOME/.erlang.cookie` zu finden.
- Unter Windows unter `%HOMEDRIVE%%HOMEPATH%\.erlang.cookie` oder `%USERPROFILE%\.erlang.cookie` abgelegt.
- Alternativ per Umgebunsvariable `RABBITMQ_SERVER_ADDITIONAL_ERL_ARGS = "-setcookie cookie-value"` setzen (unsicher).

Nodes in Cluster

Knoten können durch direkt durch DNS oder per Konfiguration, per Plugin durch AWS Instance Directory, Consul, etcd, Kubernetes oder manuell durch das CLI zu einem Cluster zusammen geführt werden.
Alle Knoten müssen für alle Knoten per DNS oder `/etc/hosts` auflösbare FQDN haben.

`rabbitmqctl join_cluster <name>@<knoten> [--ram]`
fügt den lokalen Knoten in das Cluster, bei dem <name>@<knoten> ein Mitgliedsknoten ist. Dieser Befehl funktioniert nur, wenn die Applikation gestoppt ist (siehe: `rabbitmqctl stop_app`). Durch das Einfügen wird die lokale Datenbank verworfen und die des Clusters übernomnmen. Die Option `--ram` weist den lokalen Knoten an, als RAM-Only Knoten zu arbeiten. Es wird keine lokale Datenbank persistiert.

`rabbitmqctl cluster_status`
listet alle aktuellen Knoten im Cluster auf.

`rabbitmqctl forget_cluster_node [--offline] <name>@<knoten>`
entfernt Knoten <name>@<knoten> aus dem Cluster. Der Befehl muss auf einem aktiven Knoten des Clusters ausgeführt werden. <name>@<knoten> muss offline sein. Mit dem Parameter `--offline` kann der Befehl auch auf dem Knoten eines insges. offline stehenden Clusters ausgeführt werden. Dies ist nötig, wenn dieser Knoten der letzte valide eines Clusters war, aber nicht startet, weil die anderen Knoten nicht erreichbar sind.

`rabbitmqctl update_cluster_nodes <name>@<knoten>`
aktualisiert den lokalen Knoten durch den Clusterinhalt von <name>@<knoten>. Dieser muss Teil des Clusters sein, in dem auch der lokale Knoten ist.

`rabbitmqctl force_boot`
befiehlt dem lokalen Knoten, zu starten, obwohl dieser ggf. nicht starten würde, da dieser davon ausgeht, nicht der letzte Knoten des Clusters gewesen zu sein. Damit gehen alle Änderungen in der Cluster-Datenbank verloren, die nach dem Shutdown des lokalen Knotens passiert sind.

`rabbitmq sync_queue [-p vhost] queue`
erzwingt die Synchronisation der Queue "queue", ggf. auf Vhost "vhost". Während die Synchronisation läuft, werden Producer und Consumer belockiert. der Befehl funktioniert nur auf Queues die auf mehr als einen Knoten gespiegelt sind.

`rabbitmq purge_queue [-p vhost] queue`
löscht den Inhalt einer Queue.

Knotenausfall-Behandlung

RabbitMQ ist im Clusterbetrieb empfindlich für Netzwerklatenzen oder auch kurzes Einfrieren von Knoten. Dies sollte unbedingt vermieden werden. In der Default-Konfiguration führt dieses zu einer Netzwerkpartitionierung, einer sog. "Split-Brain"-Situation. Dies kann vermieden werden. RabbitMQ kennt folgende Konfigurationsmöglichkeiten hierzu für den Wert "cluster_partition_handling". Siehe http://www.rabbitmq.com/partitions.html:

ignore
: der Default-Zustand. Knoten arbeiten weiter, obwohl sie den oder die anderen Knoten nicht mehr erreichen können. Split-Brain ist meist die Folge.

autoheal
: sorgt für eine automatische Kompensation eines kurzzeitigen Netzwerkausfall. Hierbei wird Datenverlust auf einem der Teile in Kauf genommen und mit der Majorittät weiter gearbeitet.

pause_minority
: der Knoten hält an und verweigert Verbindungen, wenn er die Hälfte oder weniger Knoten des Clusters erreichen kann. Die Knoten die Mehr als die Hälfte sehen, arbeiten weiter. Klassisches Quorum, funktioniert optimal bei einer ungeraden Anzahl an Knoten ab drei Stück. Bei zwei Knoten wird ein "Split Brain" verhindert, allerdings ist der Cluster in der Zeit der Partitionierung nicht verfügbar. Angehaltene Knoten sollten automatisch wieder in den Cluster integriert werden, sobald sie wieder verfügbar sind. In dieser Einstellung wird Konsistenz über Verfügbarkeit gestellt.

pause_if_all_down
: der Knoten hält an, wenn er keinen anderen Knoten mehr erreichen kann. Diese Einstellung sollte sinnvollerweise mit der Einstellung für `recover` und ggf. einer Knotenliste konfiguriert werden.

Diese Einstellungen sollten auf allen Knoten grundsätzlich identisch sein. Lediglich bei "pause_if_all_down" können bei Bedarf die Knotenlisten abweichen. Diese Einstellung kann nur in der RabbitMQ-Konfigdatei geändert werden und erfordern einen geordneten Neustart aller Knoten.

Betrieb

Starten und Stoppen

`rabbitmqctl start_app`
: startet RabbitMQ auf dem lokalen Knoten bei bereits laufendem Erlang-System. Wird benötigt, wenn man RabbitMQ vorher für Konfigurationsänderungen gestoppt hat.

`rabbitmqctl stop_app`
: hält den RabbitMQ-Knoten an ohne die Erlang-Runtime zu beenden oder den Knoten aus einem Cluster zu entfernen. Die anderen Clusterknoten werden über das Anhalten informiert.

`rabbitmqctl stop [pidfile]`
: beendet die Erlang-Runtime incl. RabbitMQ. Optional kann eine entsprechende PID-Datei angegeben werden. Dieser Befehl blockiert nicht.

`rabbitmqctl shutdown`
: ähnlich `stop`, jedoch blockiert die Ausführung bis der Erlangknoten herunter gefahren ist. Es muss kein PID-File im Dateisystem vorhanden sein. Exitcode$\neq$0, wenn RabbitMQ nicht lief.

`rabbitmqctl wait pid_file | rabbitmqctl wait --pid pid`
: wartet, bis die pid-Datei angelegt wurde und der Prozess mit der PID aus dem pid-File oder dem Parameter `--pid` läuft. Hilfreich für DevOps-Scripts.

`rabbitmqctl rotate_logs`
: ab Version 3.7, sorgt for Logrotation. Vor Version 3.7 über die Konfiguration vom `lager`-Plugin. Wird nicht im Zusammenhan mit Tools wie Logrotate benötigt, da dies automatisch erkannt wird.

`rabbitmqctl reset`
: setzt den lokalen Knoten zurück. Damit wird der Knoten auch aus einem Cluster entfernt. Alle Daten in der lokalen Datenbank werden gelöscht. Der Knoten muss vorher per `stop_app` gestoppt werden.

`rabbitmqctl force_reset`
: analog zu `reset`, allerdings wird der Knoten direkt zurückgesetzt, auch wenn er nicht aus einem Cluster entfernbar ist oder die lokale Datenbank defekt ist. Dieser Befehl sollte als letzter Ausweg gesehen werden.

`rabbitmqctl node_health_check`
: prüft den Knoten auf Probleme, testet ob RabbitMQ läuft und keine Alarme gesetzt sind.

Statusabfrage

`rabbitmqctl status`
: zeigt den aktuellen Status des Exchange Brokers an.

`rabbitmqctl list_queues [-p vhost] [--offline | --online | --local] [selektor ...]`
: listet Queues auf. Parameter `--offline` gibt nur nicht verfügbare Queues aus, `--online` nur verfügbare, `--local` nur die auf dem lokalen Knoten. Details zu Selektor siehe Man-Page.

`rabbitmqctl list_exchanges [-p vhost] selektor...`
: gibt Informationen zu Exchanges aus. Details zu Selektor siehe Man-Page.

`rabbitmqctl list_bindings [-p vhost] selektor...`
: gibt Informationen zu Bindungen aus. Details zu Selektor siehe Man-Page.

`rabbitmqctl list_connections [-p vhost] selektor...`
: gibt Informationen zu TCP-Verbindungen aus. Details zu Selektor siehe Man-Page.

`rabbitmqctl list_channels selektor...`
: gibt Informationen zu aktuellen Kanälen aus. Details zu Selektor siehe Man-Page.

`rabbitmqctl list_consumers`
: listet alle Konsumenten auf.

`rabbitmqctl environment`
: listet alle Umgebungsvariablen auf.

`rabbitmqctl report`
: generiert einen vollständigen Report des RabbitMQ und der Erlang-Runtime.

Benutzerverwaltung

Die folgenden Befehle ändern von der Konsole aus direkt die Usereinträge in der RabbitMQ - internen Benutzerdatenbank. Die Benutzerdatenbank wird im Clusterbetrieb repliziert. Externe Authentifizierungs- systeme wie LDAP etc. werden nicht geändert. Der Default-User und -Passwort ist `guest`. Dieser hat zunächst Admin-Rechte!

Benutzer per CLI verwalten

`rabbitmqctl list_users`
zeigt alle in RabbitMQ verwalteten Benutzer an.

`rabbitmqctl add_user <username> <password>`
legt Benutzer mit (optionalem Passwort) an.

`rabbitmqctl delete_user <username>`
löscht angegebenen User.

`rabbitmqctl change_passwort <username> <neues_pwd>`
ändert Passwort für angegeben User.

`rabbitmqctl clear_passwort <username>`
entfernt Passwort für angegeben User.

`rabbitmqctl set_user_tags <username> [tag ...]`
setzt Tag für User. "Tag" kann folgende Werte annehmen: "Admin", "Monitoring", "Management", "None" sein. Eine leere Angabe löscht alle Tags des Users.

VHosts verwalten

`rabbitmqctl add_vhost <name>`
legt einen neuen virtuellen Host-Kontext an.

`rabbitmqctl delete_vhost <name>`
entfernt einen virtuellen Host-Kontext.

`rabbitmqctl list_vhosts [<wert>]`
listet virtuelle Hosts. <wert> ist die optinale Angabe, welche Informationen über die VHosts ausgegeben werden soll. Zulässige Werte sind "name" und "tracing"

`rabbitmqctl set_permissions [-p vhost] user conf write read`
Setzt Berechtigungen für "user" auf "vhost" (default: VHost "/"). conf, write und read sind RegEx, die die Resourcen matchen, für die dem User das jeweilige Recht eingeräumt werden soll.

`rabbitmqctl clear_permissions [-p vhost] user`
entfernt die Berechtigungen für User auf dem VHost (default "/")

`rabbitmqctl list_permissions [-p vhost]`
zeigt die vergebenen Berechtigungen auf dem VHost an.

`rabbitmqctl list_user_permissions user`
zeigt alle Berechtigungebn für User an.

`rabbitmqctl set_topic_permissions conf write read`
Setzt Berechtigungen für User auf der Exchange auf VHost (default: "/"). Write und Read sind RegEx die den Routing Key der Topics matchen.

`rabbitmqctl clear_topic_permissions [-p vhost] user [exchange]`
Entfernt die User-Berechtigungen auf VHost (default: "/") auf der angegebenen Exchange oder auf allen Exchanges.

`rabbitmqctl list_topic_permissions [-p vhost]`
Zeigt die vergebenen Topic Permissions auf VHost.

Crash Recovery

Im Falle eines fatalen Ausfalls eines Clusters oder Einzel-Systems kann es passieren, dass die interne datenbank defekt ist. In den meisten Fällen kann die Erlang-Runtime die eingesetzte Mnesia-Datenbank selbstständig reparieren. Sollte es hier dennoch probleme geben, hilft möglicherweise dieses Kapitel weiter: http://erlang.org/doc/apps/mnesia/Mnesia_chap7.html#id85020
Warnung: Grundsätzlich sollte jedoch in diesem Fall vor Experimenten vom Offline-System eine Sicherungskopie bzw. vorhanden sein oder ggf. ein Experte hinzugezogen werden.

Updates

Solobetrieb

Updates eines einzelnen RabbitMQ, der nicht im Cluster-Betrieb arbeitet, können in-place durchgeführt werden, wenn der RabbitMQ nicht läuft. Hierbei sollten keine großen Versionssprünge ausgeführt werden. Ein Versionsschema von <major.minor.patchlevel> angelegt bedeutet dies, dass Patchlevel in aller Regel beliebig installiert werden können, bei Minor- und Major-Versions-wechseln aber auf jeden Fall mit Vorsicht vorgegangen werden muss. Bei Minor- oder Major-Versionswechseln sollte sicherheitshalber keine Minor-Version ausgelassen werden. Hierzu geben die Changelogs entsprechend Auskunft.

Ferner ist ein Cold-Backup des Systems obligatorisch. Zu Beachten ist auch, dass zum RabbitMQ auch ggf. die passende Erlang/OTP-Version mit aktualisiert werden muss.

Wichtig: Keinesfalls sollte man einfach ein Paket mit neuerer Version aus einem Upstream-Repository über die bestehende Installation installieren. Bei Serverneustarts ist zu beachten, dass nicht-persistente Queues und Messages verloren gehen.

Mehr Informationen siehe
https://www.rabbitmq.com/upgrade.html

Clusterbetrieb

Im Clusterbetrieb können nur Patchlevel (s.o.) problemlos geupdatet werden. Clusterupgrades können wahlweise über ein Zurückbauen des Clusters auf einen einzelnen Knoten - also zurückbauen auf Solobetrieb - und Update wie im Solobetrieb mit anschließendem Ergänzen neuer, aktueller und leerer Knoten zum Cluster erfolgen. Rolling Upgrades sind zumeist nicht zielführend.

Alternativ kann ein Blue-Green-Deployment mit Neubau eines Clusters neuerer Version und Übertragung der Daten per Federation-Plugin oder Shovel-Plugin von Alt nach Neu erfolgen. Mehr hierzu siehe:
https://content.pivotal.io/rabbitmq/blue-green-application-deployments-with-rabbitmq