TON OS DApp Server Devops Contest [3.08.-3.09.2020] SUBMISSION

Repository: https://github.com/samostrovskyi/TON-OS-DApp-Server-deploy
Wallet address: 0:a2c66fbd01f0193c39127d1dd825e6d144d0581ca82a72a747d0af343b2c0b0b

What

We are ready to announce TON OS DApp Server management tool which follow all the nowadays best practices:

- Fully-automated approach
- Centralized management of remote servers
- Distributed microservice architecture
- Centralized logging and monitoring
- Alerting and notifications
- Security basics

Why

The main idea of this project is to provide an ability to manage TON OS DApp Server setup as simply as possible for development and production purposes. Now you can easily install or upgrade TON OS DApp Server by single command based on Ansible.

Who

We are a team of 2 DevOps professionals who are passionate about blockchain technologies and wanna bring value to FreeTON development. Feel free to contact us in case of any questions:

Telegram	Github	Forum.freeton.org	Gmail
@renatSK @sostrovskyi	@ddzsh @samostrovskyi	@Gofman @sostrovskyi	senegalelastico@gmail.com renatskitsan@gmail.com

Technology stack

- **Ansible** - tool which automates all configuration
- **Docker** - builds images and hosts TON-OS-DApp-Server containers
- **Gelf** - logging driver is a convenient format that is understood by a number of tools
- **Docker-compose** - describes TON-OS-DApp-Server
- **ELK** - collects logs from all the Docker containers
- **Prometheus** - scrapes metrics from TON-OS-DApp-Server components
- **Prom-exporters** - expose metrics which prometheus collect (q-server,ton-node,arangodb,etc)
- **Alertmanager** - responsible for alerts and notifications
- **Alertmanager-bot** - application which receive web-hook and send alerts to Telegram

System requirements

Configuration	CPU (cores)	RAM (GiB)	Storage (GiB)	Network (Gbit/s)
Recommended	24	192	2000	1

NOTE: SSD disks are recommended for the storage.

Prerequisites

1) At least 1 server with public IP address (at least 3 recommended)
2) Public DNS A entries for DApp server[1]:
    - Graphql A record
    - ArangoDB A record
    - ArangoDBNI A record

3) Ansible installed. Required version - 2.9.6

If you don't have a public DNS zone you can buy from any provider. e.q. GoDaddy. If you don't familiar with that just follow this page: https://ua.godaddy.com/domains

[1] In case of setup **single-node architecture** you can point A records to single IP

As an additional layer of security you can add domain to Cloudflare (or use letsencrypt, playbook by default will issue certificates in letsencrypt automatically)
By default playbook will install 2 endpoints with basic auth (see group_vars/proxy):
  <inventory_hostname>
  arangodb.<inventory_hostname>
  arangodbni.<inventory_hostname>
If you want to customize this hostnames specify ARANGODB_VIRTUAL_HOST and ARANGODBNI_VIRTUAL_HOST variables inside group_vars/arangodb.

Quick start

1) Navigate to ansible directory:
   cd ansible

2) Specify where to install DApp server components in hosts file:
   vim hosts[2]
3) Copy ssh pub key to remote server (optional, because password also can be used)[3]
4) Install docker, docker-compose, and setup docker network:
   ansible-playbook -u root -i hosts run.yml -t install

5) Run DApp server node:
   ansible-playbook -u root -i hosts run.yml -t up

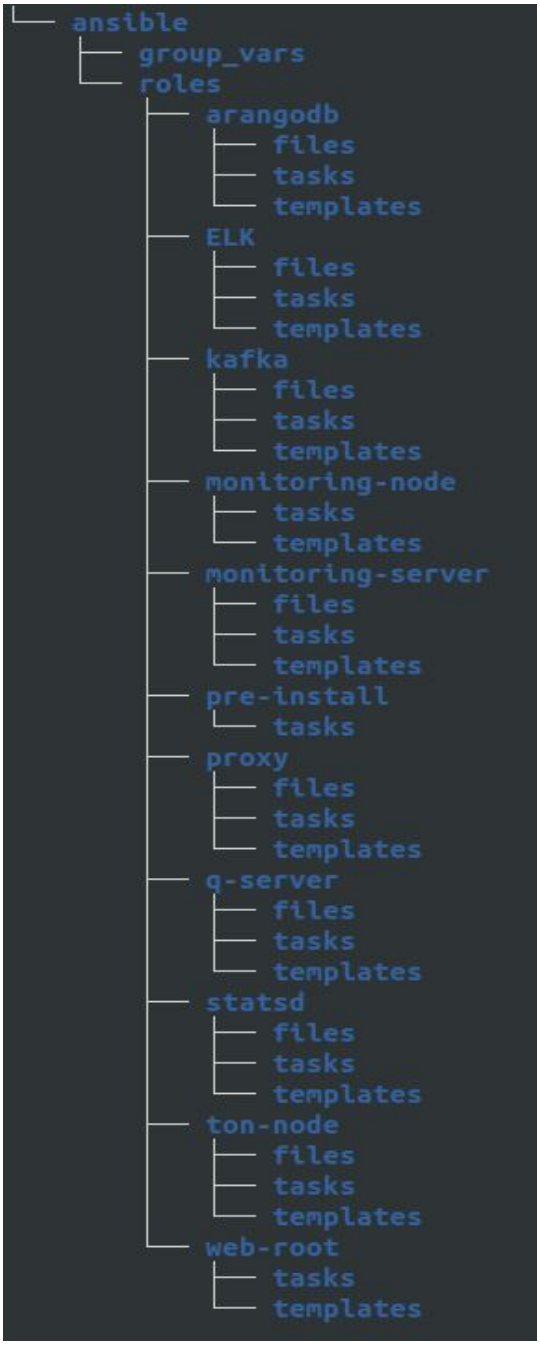
6) Stop DApp server node:
   ansible-playbook -u root -i hosts run.yml -t down

Project structure

Project is based on Ansible and follows all the best practices for writing playbooks.

---

[2] Please note that only hostnames should be used!
[3] Alternatively you can use ansible_connection=local
[proxy]
<hostname> ansible_connection=local

Ansible project structure	Description
	Ansible project includes all the required roles to spin up monitoring, logging, alerting and DApp server nodes itself. Each component has its own role. Components of DApp server node are the following: 1) Arangodb 2) Kafka 3) Proxy 4) Q-server 5) Statsd 6) Ton-node 7) Web.root Monitoring, logging and alerting stack: 1) ELK 2) Monitoring-node 3) Monitoring-server And pre_install role to manage docker setup. Each role has variables defined in *group_vars/<ansible_role_name>* which are used to parse “Tasks” and “Templates”. Each role also has several directories: 1) *Files* (build files and static configs) 2) *Tasks* (list of commands to run) 3) *Templates* (docker-compose.yml and some additional configs to parse) The main file is called *run.yml* and it works in conjunction with the hosts file to manage where to execute specific roles.

Variables description

Parameter	Description	Default
ARANGO_NO_AUTH	Disabling arangodb authentication. Need to set to 0 for production	1
VIRTUAL_HOST	External hostname to access arangodb UI	arango.baremetal01.fra02.ibm.cloud
VIRTUAL_PORT		8529
LETSENCRYPT_HOST	Hostname on which SSL should be issued	arango.baremetal01.fra02.ibm.cloud
LETSENCRYPT_EMAIL		senegalelastico@gmail.com
ARANGODB_OVERRIDE_DETECTED_TOTAL_MEMORY		343579738368
ARANGO_ROOT_PASSWORD		
NETWORK_TYPE	TON Network	net.ton.dev
INSTANCE_NAME		first
ARANGO_SERVICE_SERVICE_HOST		arangodb
ARANGO_SERVICE_SERVICE_PORT		8529
ARANGONI_SERVICE_SERVICE_HOST		arangodbni
ARANGONI_SERVICE_SERVICE_PORT		8529
ARANGONI_NO_AUTH	Disabling arangodb authentication. Need to set to 0 for production	1

ARANGONI_VIRTUAL_HOST	External hostname to access arangodb webui	arangoni.baremetal01.fra02.ibm.cloud
ARANGONI_VIRTUAL_PORT	Internal ports	8529
ARANGONI_LETSENCRYPT_HOST	External hostname to access arangodb webui	arangoni.baremetal01.fra02.ibm.cloud
ARANGONI_LETSENCRYPT_EMAIL		senegalelastico@gmail.com
discovery_type	ELK	single-node
ZOOKEEPER_CLIENT_PORT		2181
ZOOKEEPER_TICK_TIME		2000
KAFKA_BROKER_ID		1
KAFKA_ZOOKEEPER_CONNECT	URL to zookeeper	zookeeper:2181
KAFKA_ADVERTISED_LISTENERS		PLAINTEXT://kafka:29092,PLAINTEXT_HOST://kafka:9092
KAFKA_LISTENER_SECURITY_PROTOCOL_MAP		PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
KAFKA_INTER_BROKER_LISTENER_NAME		PLAINTEXT
KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR		1
KAFKA_JMX_PORT		9581
KAFKA_LOG_RETENTION_HOURS		4
KAFKA_LOG_ROLL_MS		600000
KAFKA_LOG_SEGMENT_BYTES		1073741824
KAFKA_LOG_RETENTION_CHECK_INTERVAL_MS		300000
KAFKA_CLEANUP_POLICY		delete

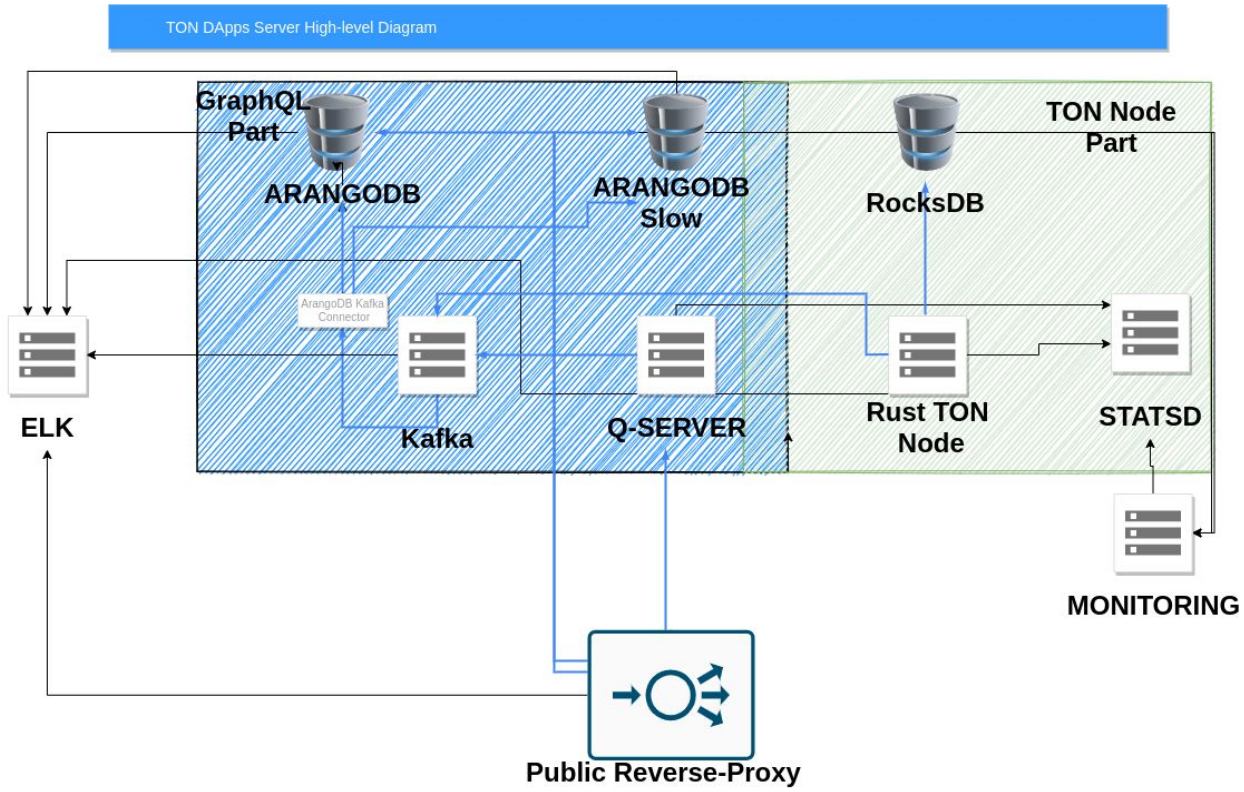
KAFKA_RETENTION_MS		43200000
KAFKA_MESSAGE_MAX_BYTES		3001000
KAFKA_RECEIVE_MESSAGE_MAX_BYTES		3001000
KAFKA_REPLICA_FETCH_MAX_BYTES		3001000
CONNECT_BOOTSTRAP_SERVERS		kafka:29092
CONNECT_REST_ADVERTISED_HOST_NAME		connect
CONNECT_REST_PORT		8083
CONNECT_GROUP_ID		compose-connect-group
CONNECT_CONFIG_STORAGE_TOPIC		docker-connect-configs
CONNECT_CONFIG_STORAGE_REPLICATION_FACTOR		1
CONNECT_OFFSET_FLUSH_INTERVAL_MS		10000
CONNECT_OFFSET_STORAGE_TOPIC		docker-connect-offsets
CONNECT_OFFSET_STORAGE_REPLICATION_FACTOR		1
CONNECT_STATUS_STORAGE_TOPIC		docker-connect-status
CONNECT_STATUS_STORAGE_REPLICATION_FACTOR		1
CONNECT_KEY_CONVERTER		org.apache.kafka.connect.storage.StringConverter
CONNECT_VALUE_CONVERTER		io.confluent.connect.avro.AvroConverter

CONNECT_VALUE_CONVERTER_SCHEMA_REGISTRY_URL		http://schema-registry:8081
CONNECT_INTERNAL_KEY_CONVERTER		org.apache.kafka.connect.json.JsonConverter
CONNECT_INTERNAL_VALUE_CONVERTER		org.apache.kafka.connect.json.JsonConverter
CONNECT_ZOOKEEPER_CONNECT		zookeeper:2181
CLASSPATH		/usr/share/java/monitoring-interceptors/monitoring-interceptors-5.2.1.jar
CONNECT_PRODUCER_INTERCEPTOR_CLASSES		io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor
CONNECT_CONSUMER_INTERCEPTOR_CLASSES		io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor
CONNECT_PLUGIN_PATH		/usr/share/java,/usr/share/confluent-hub-components
CONNECT_LOG4J_LOGGERS		org.apache.zookeeper=ERROR,org.I0Itec.zkclient=ERROR,org.reflections=ERROR
CONNECT_KAFKA_JMX_PORT		9584
CONNECT_FETCH_MESSAGE_MAX_BYTES	Important limit! Without it ton-node can’t work	4000000
CONNECT_MAX_REQUEST_SIZE	Important limit! Without it ton-node can’t work	4000000
CONNECT_MAX_PARTITION_FETCH_BYTES	Important limit! Without it ton-node can’t work	4000000
SCHEMA_REGISTRY_HOST_NAME		schema-registry
SCHEMA_REGISTRY_KAFKASTORE_CONNECTION_URL		zookeeper:2181

SCHEMA_REGISTRY_JMX_PORT		9582
ADVERTISED_LISTENER		
telegram_user_ids	List of TG IDs Check @getmyid_bot	
telegram_bot_token	TG Bot token Check @BotFather	
grafana_username	Grafana UI user	
grafana_password	Grafana UI password	
auth	Boolean If true basic auth will be configured for each endpoint	
username	Proxy auth user	
password	Proxy auth password	
version	Q-server git branch to build	master
VIRTUAL_HOST		q-server
VIRTUAL_PORT		4000
Q_DATABASE_SERVER		arangodb:8529
Q_SLOW_DATABASE_SERVER		arangodbni:8529
Q_REQUESTS_MODE		kafka
Q_REQUESTS_SERVER		kafka:9092
Q_REQUESTS_TOPIC		requests
Q_DATABASE_MAX_SOCKETS		100
Q_SLOW_DATABASE_MAX_SOCKETS		20
IMAGE	Statsd docker image repo	prom/statsd-exporter:v0.12.2
ARGS		--statsd.mapping-config=/statsd-mappings/statsd-mapping.yaml
UDP_PORT		9125
TCP_PORT		9102

IntIP		30303
version	DApp node git branch	master
ADNL_PORT		30303
VALIDATOR_NAME		my_validator
NETWORK_TYPE		net.ton.dev
CONFIGS_PATH		/ton-node/configs
STATSD_DOMAIN		statsd:
STATSD_PORT		9125
MEM_LIMIT		32G
STAKE		
MSIG_ENABLE		
SDK_URL		
protocol	Web.root protocol	http

High-Level Architecture diagram



TON OS DApp Server is a set of services enabling you to work with TON blockchain.

The core element of TON OS DApp Server is a TON node written in Rust focused on performance and safety. TON OS DApp Server provides a set of services serving TON SDK endpoint: scalable multi-model database ArangoDB with the information about all blockchain entities (like accounts, blocks, transactions, etc.) stored over time, distributed high-throughput, low-latency streaming platform Kafka, TON GraphQL Server (aka Q-Server) for serving GraphQL queries to the database and Nginx web-server.

Q-Server is accessible with GraphQL HTTP/WebSocket protocol on port "4000" and path "/graphql". Q-Server represents the read only part to provide GraphQL Endpoint. In case of write operation Q-Server creates an event into the Kafka and responds back. Then ton-node will catch this message and send it into blockchain.

Rust TON Node works with Kafka and two arangodb databases and also depends on statsd. As a database TON Node uses RocksDB.

With help of this ansible-playbook TON DApps Server can be setup in several ways:

- Single-node architecture

In this case all components will be installed on a single node (“all-in-one”). Including monitoring server and logging server.

- Multi-node architecture

Automation scripts provide a way to install each component to different nodes. We recommend installing kafka, arangodb`s, q-server separately from tone node. To ensure that the ton-node will have maximum possible network throughput and IOPS.

Kafka used by q-server and ton-node. Also Kafka uses two Arangodb databases. For Kafka -> Arangodb communication kafka uses *arangodb kafka connector.*

Logging

Please, note that for production use it’s recommended to setup logging software without public access or additionally configure authentication.

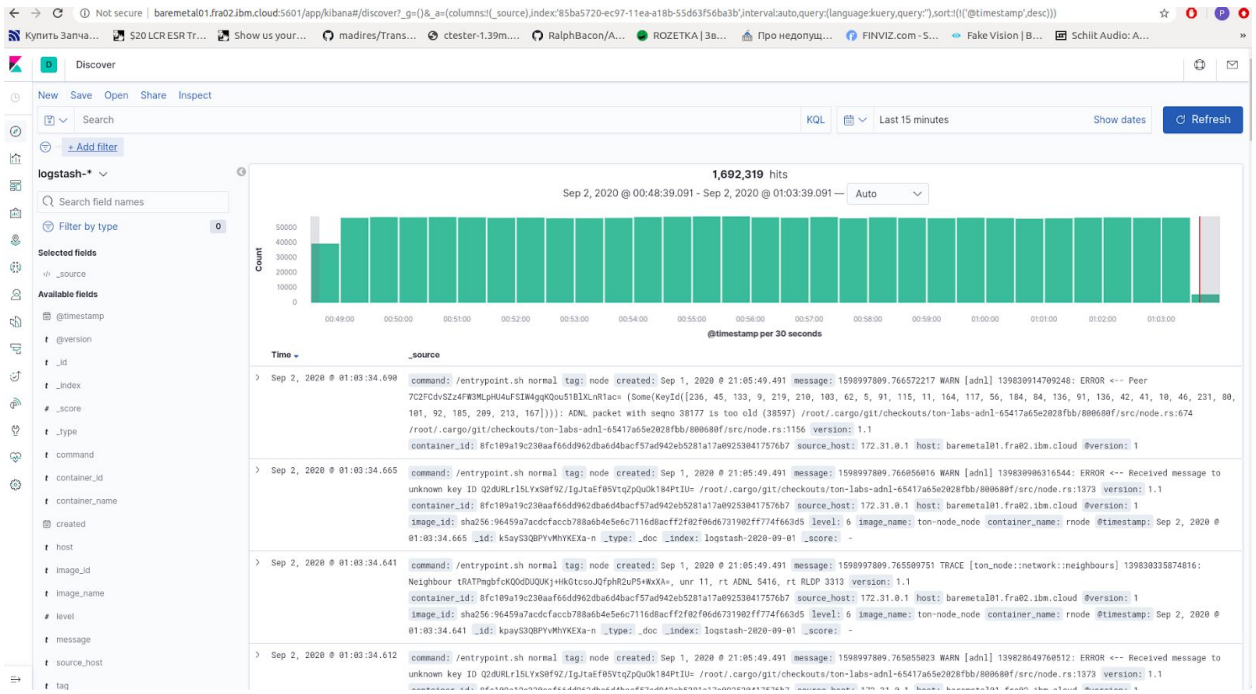
Each docker container has logging configuration to send logs over UDP to remote (or local in case of single node architecture). All processes inside docker containers configured to send logs to stdout/stderr. This way helps ensure that components (especially **ton-node**) will have a maximum of available server IOPS. Automation scripts use a Gelf logging driver.
Configuration example:

```
  logging:
    driver: gelf
    options:
      gelf-address: "udp://logstash:12201"
      tag: "statsd"
```

Logs go to the logstash docker container. In logstash we can configure additional logs parsing and processing. From logstash logs passes to elasticsearch. Kibana setup to provide a web interface for logs representation and search.
Logs from each container tagget get a unique tag to provide the ability to separate logs. There are several tags exists:

- ton-node
- q-server
- arangodb
- kafka
- statsd, etc ...

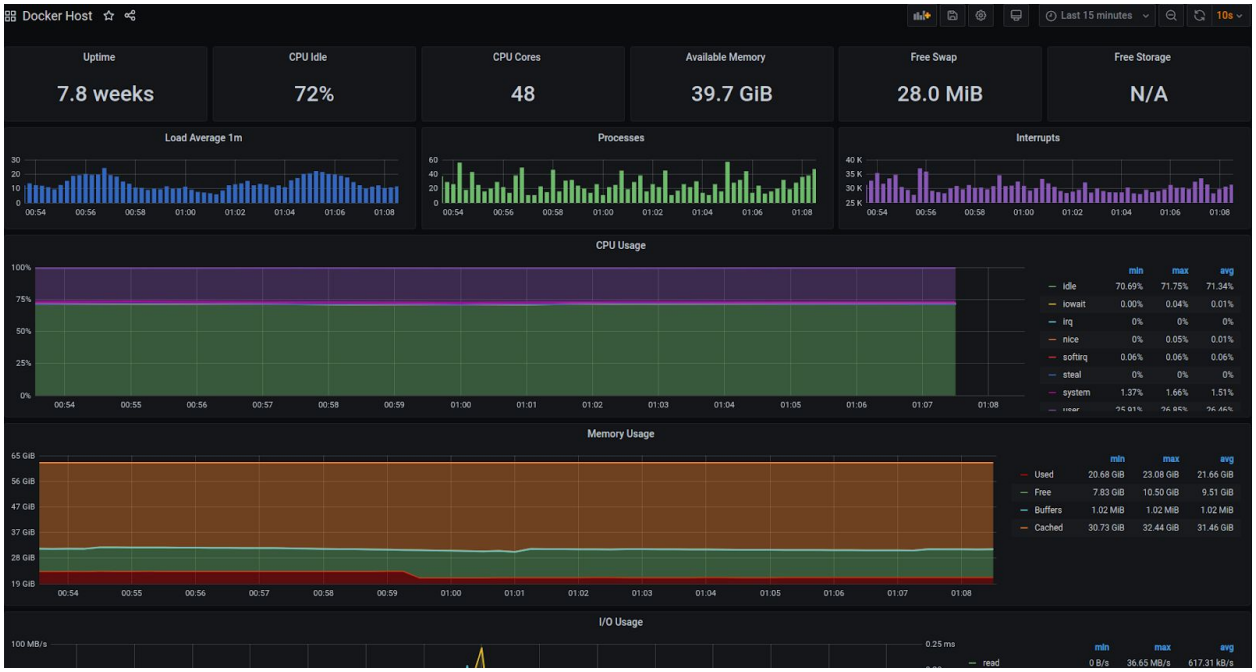
Kibana:



Monitoring

General prometheus dashboard

Alerts, monitoring stack information



Docker containers:

CPU/Memory/Disk/Status

Docker Host Dashboard

Overall information about docker hosts. CPU/Memory/IOPS, etc

Q-Server configured to reports several StatsD metrics:

Metric	Type	Tags	Description
qserver.start	counter	collection=name	Incremented for each start. Contains the `version` tag.
qserver.doc.count	counter	collection=name	Incremented for each db event (document insert/update)
qserver.post.count	counter		Incremented for each node request
qserver.post.failed	counter		Incremented for each failed node request
qserver.query.count	counter	collection=name	Incremented for each db query
qserver.query.failed	counter	collection=name	Incremented for each failed db query
qserver.query.slow	counter	collection=name	Incremented each time when q-server has encountered slow query
qserver.query.time	timer	collection=name	Reported each time when q-server has finished query handler from SDK client
qserver.query.active	gauge	collection=name	Updated each time when q-server has started and finished query handler from SDK client

Rust TON Node configured to reports this StatsD metrics:

Metric	Type	Tags	Description
rnode.shards_client_mc_block	gauge		Reportes client shard block
node.last_applied_mc_block	gauge		Reportes applied block in current block

We have aggregated all information related to q-server and rust ton node to Grafana dashboard:

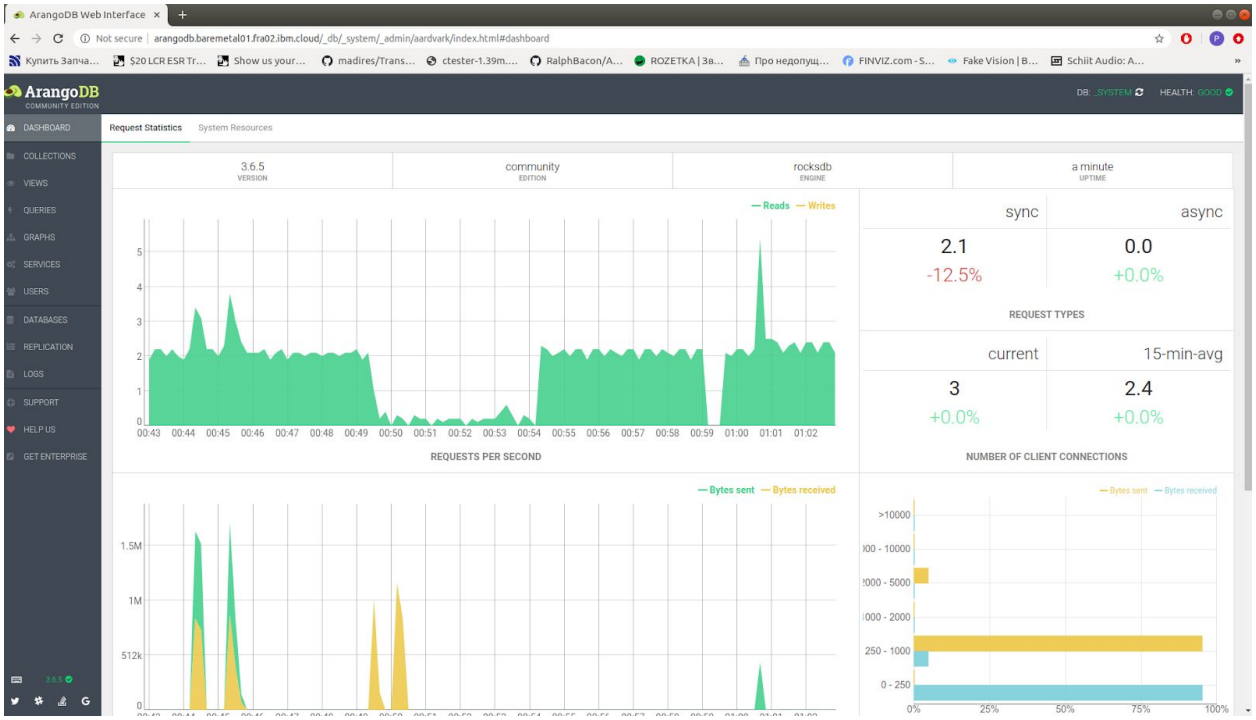
ArangoDB configured to provide this metrics to Prometheus:

Metric	Type	Tags	Description
arangodb_process_statistics_system_time	gauge	collection=name	Amount of time that this process has been scheduled in user mode, measured in seconds.
arangodb_process_statistics_user_time	279.03		
arangodb_server_statistics_physical_memory	3.43579738368e+11	gauge	Number of seconds elapsed since server start.
arangodb_server_statistics_server_uptime	gauge		
arangodb_client_user_connection_statistics_request_time_sum	gauge		
arangodb_client_user_connection_statistics_request_time_sum	gauge		
arangodb_client_user_connection_statistics_total_time_bucket	Total time needed to answer a request (only user traffic)		

…

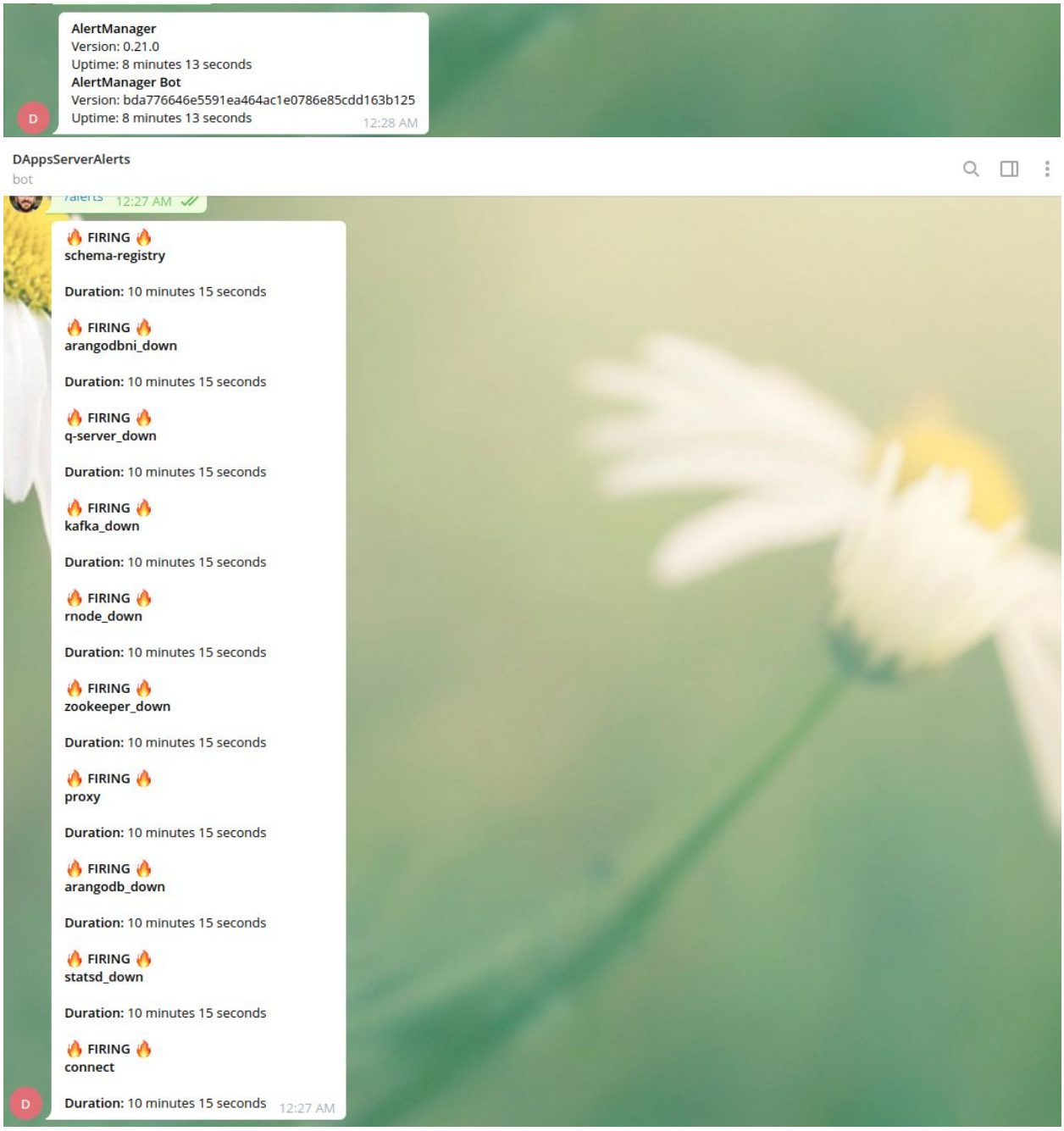
Metrics aggregated to Grafana Dashboard:

Also optionally arangodb web interface could be exposed:



Alerts

We configured Telegram bot to send notification to Telegram. Bot added to prometheus docker-compose. Prometheus configured to send a web-hook to it. To create bot use @botfather the provide token into vars and start monitoring-server playbook
Alerts example:



GraphQL query examples

After DApss Server installed. We need to verify that DApss Server works properly. We need to check both types of queries **read** and **write**. Some examples how to do that described below:
Queries to **read** data from blockchain**:**

```
{
  accounts(
    filter: {
```

```
    id: {
      eq: "0:5b168970a9c63dd5c42a6afbcf706ef652476bb8960a22e1d8a2ad148e60c0ea"
    }
   }
  ) {
   id
   balance
   acc_type
   code
  }
}
```

This query will work only if the ton-node has successfully started the synchronization process:

```
query{
  blocks(filter:{
    workchain_id:{
      eq:-1
    }
  }
    orderBy:{
      path:"seq_no"
      direction:DESC
    }
    limit: 1
  )
  {
    master{
      shard_hashes{
        shard
      }
    }
  }
}
```

The easiest way to check **write** operation to TON blockchain though DApps Server is to use tonos-cli.

TONOS CLI examples

To configure tonos-cli to use your DApps Server endpoint you can use this command:
tonos-cli config --url http<s>://<hostname>
This command will create tonlabs-cli.conf.json in the current directory. Then you are ready to submit transactions.

tonos-cli call 0:a8781e344d996d8f6c6bb2a742530b097d8f92bfff1f7624be6a1eadcbd0cccb

submitTransaction '{"dest":"0:b432d62676c97fbb10094cdfe167d29cb5b1fbc76de1e8c48b2ee1fc5ae6cafb","value":1000000000,"bounce":"false","allBalance":"false","payload":""}' --abi SafeMultisigWallet.abi.json --sign deploy.keys.json

Maintainers

Telegram:
@renatSK
@sostrovskyi

Github:
@ddzsh
@samostrovskyi

Forum.freeton.org:
@Gofman
@sostrovskyi

Gmail:
senegalelastico@gmail.com
renatskitsan@gmail.com

FreeTON wallet address:
0:a2c66fbd01f0193c39127d1dd825e6d144d0581ca82a72a747d0af343b2c0b0b
Feel free to donate some crystals :) (of course if you like our scripts and support)

Feature plans

- Extend metrics and grafana ton-node dashboard
- Add logstash configuration to parse each type of logs with special rules
- Add elasticsearch logs rotation
- Add basic auth for Kibana
- Add possibility to deploy High Availability components (kafka cluster, zookeeper cluster, arangodb in HA mode, multiple TON-nodes, HA reverse proxy endpoint)