High Level RMTools Architecture

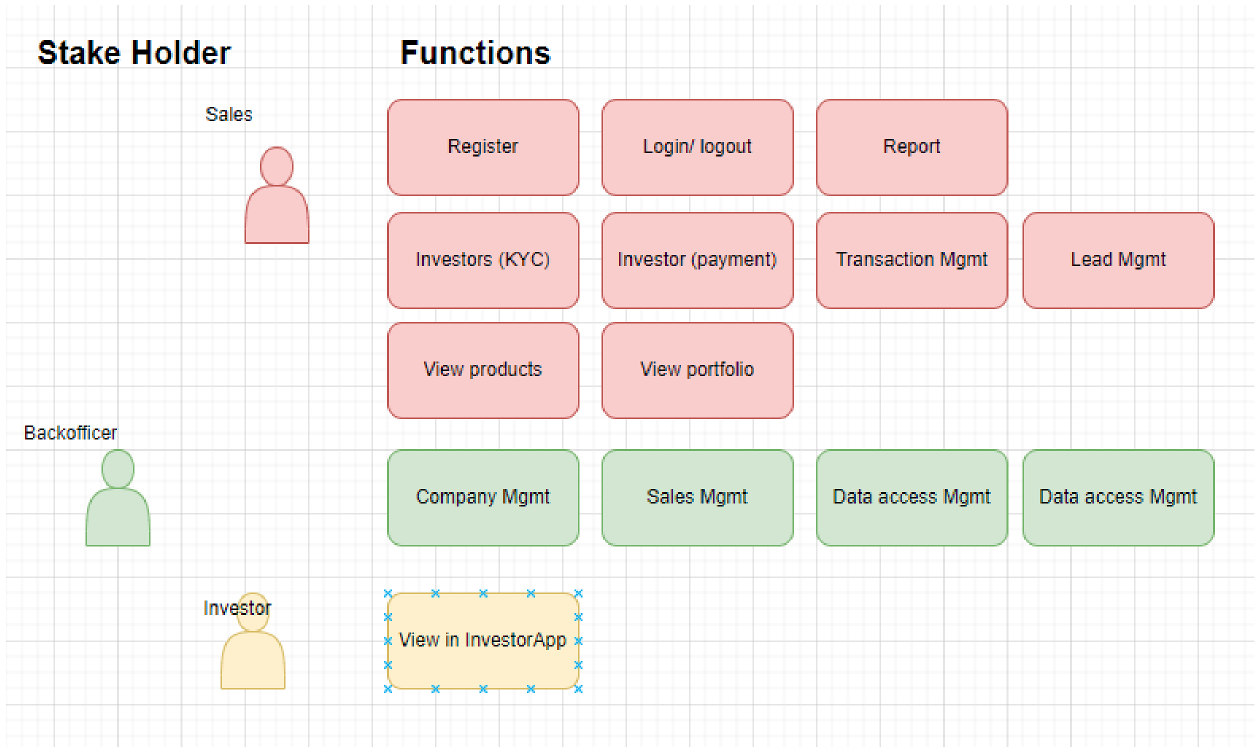
Version	Author	Note
0.1.0	pthoang@mail.com	2022-04-01: Stub information.
1.0.0	pthoang@mail.com	2022-06-25: Remove approaching solutions (akamai, jeager)
1.1.0	pthoang@mail.com	2022-07-06: Add + swagger link (API docs) + description for trigger function in db
1.2.0	pthoang@mail.com	2022-07-07: Add + assumption + system limit

Background

- Write an RMTool for sales (mobile app) and back-officer (web app), so that sales can use this as portable POS to consult to investors.
- The system needs to integrate with other BiB apps & 3rd parties services: notification (FCM, APN), pubsub (GCP) and BIB central investor/transaction management.
- The data should be inconsistent across apps in Bib: BibMoney, BibInvestor.

Requirement

- Need to fit Bib unique business, very specific flow and regulatory compliance needs.
- Support multi-tenants / multi-companies.



Note: Investor can use some link for approve transaction, remote registration

Assumption

- RMTools is an addon to BMoney, so that RMTools should run independently and cache as much as possible, so that no need to call Bib API extensively.
- The Authen/Author are not sync to any central IAM.
- The RM Users should be independent and only the source of truth of RMTools.
- The Investors and Transactions in RMTools are lead_records and should sync to Bib System for central management.
- Lead (Investor) and Instruction (Transaction) only store in RMTools like RMUser.
- The backend (API) run on K8S/docker, so that it can be elastic roll-out when deploying and have crash, or high bandwidth increase.
- Not too much concurrent users so backend (API) can query from RDBMS/ PostgresQL. We do not apply any High Availability design (fast DB, load balancing, distributed)

System Limit Information

- ~500 concurrent users can use app (now only limit in PostgresQL database, because it's not elastic / auto-span)
- Do not have much security protection in app, so should use firewall to protect from DDOS.

Granular Detail

- Do not implement disaster recovery so all modules are important.

High level solution proposal

For backend (Spring boot)

+ Using monorepo + multi-uservice (don't use monolithic due to multiple team develop)
+ Using docker + k8s on GCP
+ Security:
    - For private API: use JWT
    - For public API: use CSRF Token (login, register, forgot)
    - Firewall: use GCP WAF + Kong/Nginx WAF (~~can consider CloudFlare~~)
    - Docker: distroless / alpine.
    - Encrypt secret (Vault Manager: Postgres Password) + hardening UUID (userID, ottID)
+ Performance:
    - Use cache (redis) for fast output cache.
    - Use Kong/Nginx cache for html cache
    - Use APM tool (DataDog)
    - Indexing at Postgresql
    - Elastic scaling at k8s
+ Logging:
    - Use Grafana/Logstash for getting logs from std.err
    - Use log4j
+ Tracking: N/A

For web (Reactjs/Static Web)

+ Using monolithic + multi-packages.
+ Also using docker + k8s on GCP.
+ Security: use browser security.
+ Performance:
    - Tree-shaking bundle optimization
    - Component render: collocation avoiding.
+ Logging:
    - Use Sentry
+ Tracking: N/A

For mobile (Fluter)

+ Using monolithic + multi-packages (SDK for each micro-services)
+ App delivery: Apple Store + Google Play
+ Security: use iOS / android security layer.
+ Performance:
    - Tree-shaking bundle optimization.
    - Follow best practices for mobile app development.
    - Caching at local storage.
+ Logging:
    - Use Firebase Crashlytic.
+ Tracking: N/A

Software stack

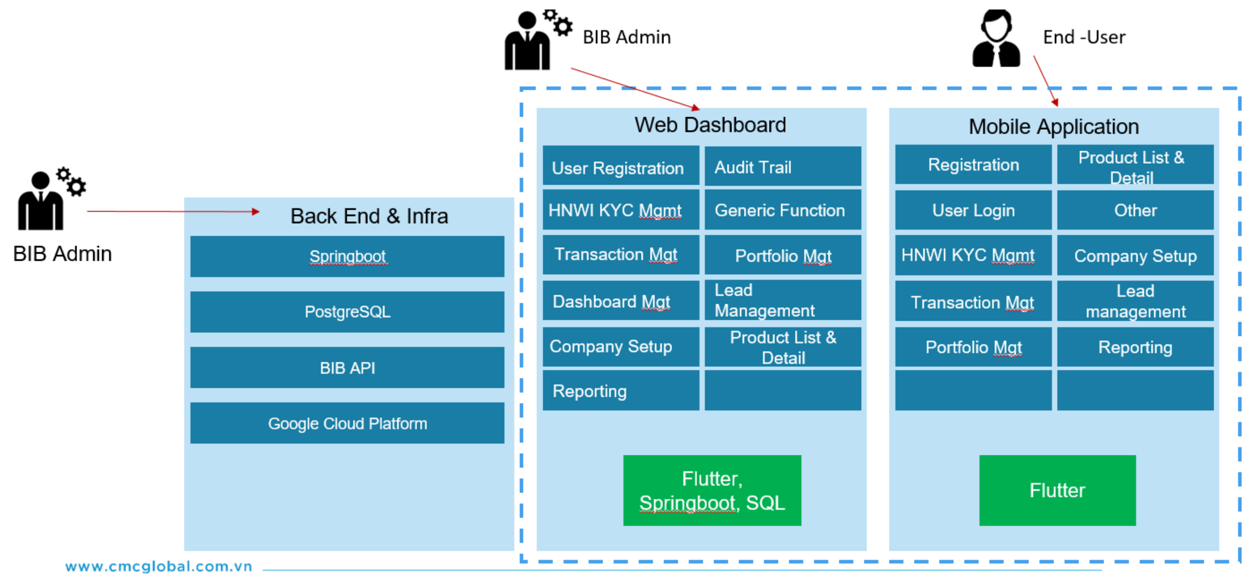


Figure 1:

App	Description:	Dependencies
Backend API (RMTool for web & mobile)	Deploy: GCP (refer: More) **Java spring boot framework** + Plugin: Spring-gcp, redis-spring, jpa. + JUnit / Jacoco + Lombok	Postgres Redis GCP (storage) Kafka (email/sms) Pubsub (real time sync) FCM (push notif)
Frontend Web (RMTools Admin)	Deploy: GCP (refer: More) **ReactJS** + Antd UI Toolkit + Axious + Jest	Backend API
Frontend Mobile (RMTool POS)	Deploy: iOS & Android (refer: More) Flutter + Plugin: picker, bloc + Flutter UI Toolkit + Flutter UI/Unit Test	Backend API
3rdparty provider	1 provider: + Google firebase + Google cloud platform	

Bib provider	2 providers:
	+ partner:
	`https://api.preproduction.bukalapak.com/_partners/`
	+ internal:
	`https://api.preproduction.bukainvestasi.com/_internal`	

Backend Architecture

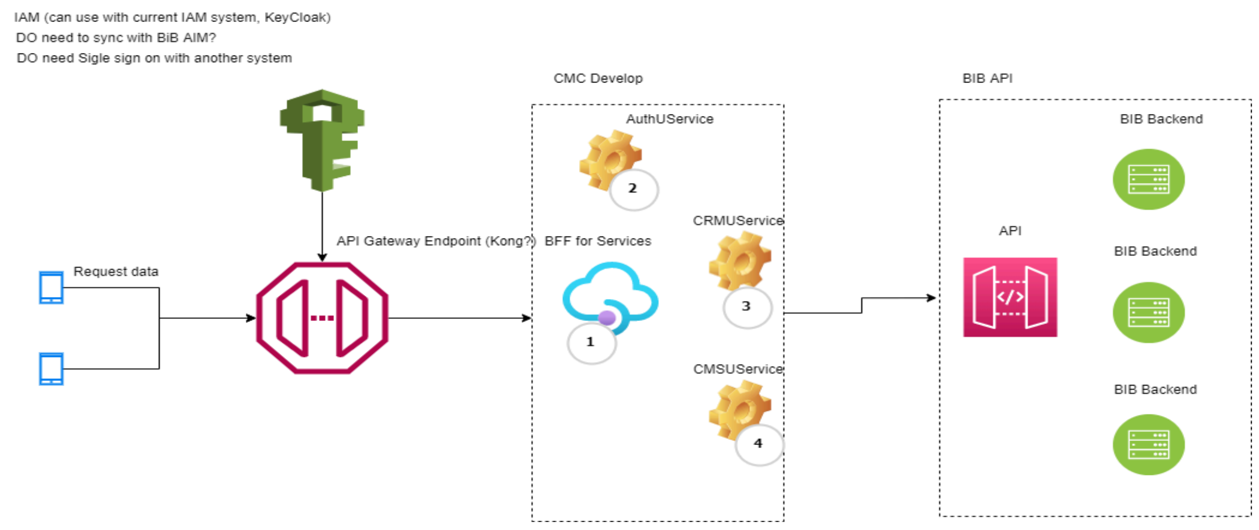

IAM (can use with current IAM system, KeyCloak)
DO need to sync with BiB AIM?
DO need Sigle sign on with another system
CMC Develop
BIB API
AuthUService
2
BIB Backend
CRMUService
3
API
BIB Backend
API Gateway Endpoint (Kong?)
BFF for Services
Request data
1
CMSUService
4
BIB Backend

Figure 1: Microservice for backend

<u>µService detail:</u>

Modules	Description:	Dependencies
Auth µService	**Dummy Service:** + Manage the user' (agents & admins) information & also user device (push device_id, OTT) + Audit trail for user actions. + Manage authentication (session, JWT) & authorization information (permission: using ACL)	N/A
CRM µService	**Dummy Service:** + Manage investors initial data (kyc for agent, CRUD for admin) + Manage transaction' investors.	N/A
BFF (Backend for Frontend)	**Smart Proxy:** + Orchestrator for system: public API (login, logout), private API (admin, RMtool: manage users, investor, txn & provide data: report, product) + Caching data in fast database for fast response. + Proxy CMS from Bib: product, portfolio + Proxy Report from Bib: transaction, portfolio	Auth µService CRM µService Shared Module
Shared module	**Libs:** + Integrated with Kafka for sending SMS/Email + Upload file to GCP	
Bib libs	Kafka SMS/Email Client.	

<u>API details (Endpoint: https://api-rmtools.preproduction.bmoney.id/swagger-ui/index.html)</u>

External API

API	Description:	Type
POST /auth/login	Login (both RMTools & Admin)	Public API

GET /auth/logout	Logout (both RMTools & Admin)	Public API
POST /auth/register	Register (only RMTools)	Public API
POST /auth/reset-password	Reset password (both)	Public API
POST /auth/verify-otp	Verify token for actions: register, new device 2FA, reset pwd, change phone_nr (only RMTools)	Public API
POST /auth/verify-unique	Check unique in db: email, phone (both)	Public API
GET /auth/companies	Get companies/tenants in RMTools (only active)	Public API
PUT /auth/change-setting	Change password, phone, email (both)	Private API/Personal
POST /auth/refresh-token	Get new JWT (both)	Private API/Personal
GET /auth/companyIds	Get all companyIds that RMAgents can access data from it (both)	Private API/Personal
GET /auth/notifications	Get all notifications	Private API/Personal
POST /auth/notifications/unsub	Unsubscribe push-notification	Private API/Personal
***** /idty/users**	CRUD for both RMAgent & Admin	Private API/Manage
PATCH /idty/users/{id}/approveOrReject	Approve RMAgent	Private API/Manage
PATCH /idty/users/{id}/activeOrDe	Active/deactive Admin	Private API/Manage
***** /crm/leads**	CRUD for lead	Private API/Personal
GET /crm/investors	View for both	Private API/Manage
PATCH /crm/investors	Approve/reject (Admin)	Private API/Manage
POST /crm/investors	Investor KYC	Private API/Personal
PUT /crm/investor/{id}/externalId	Update by external Id	Private API/Manage
POST /crm/investor/sync-bib	Sync bib	Private API/Manage
GET /crm/instructions	View for both	Private API/Manage
POST /crm/instructions	Create txn for investor	Private API/Manage
PATCH /crm/instructions/signature	Update signature from email	Public API.
GET /crm/banks	Master data	Public API
GET /crm/locations	Master data: province, city, district	Public API

Schema details:

Auth µService

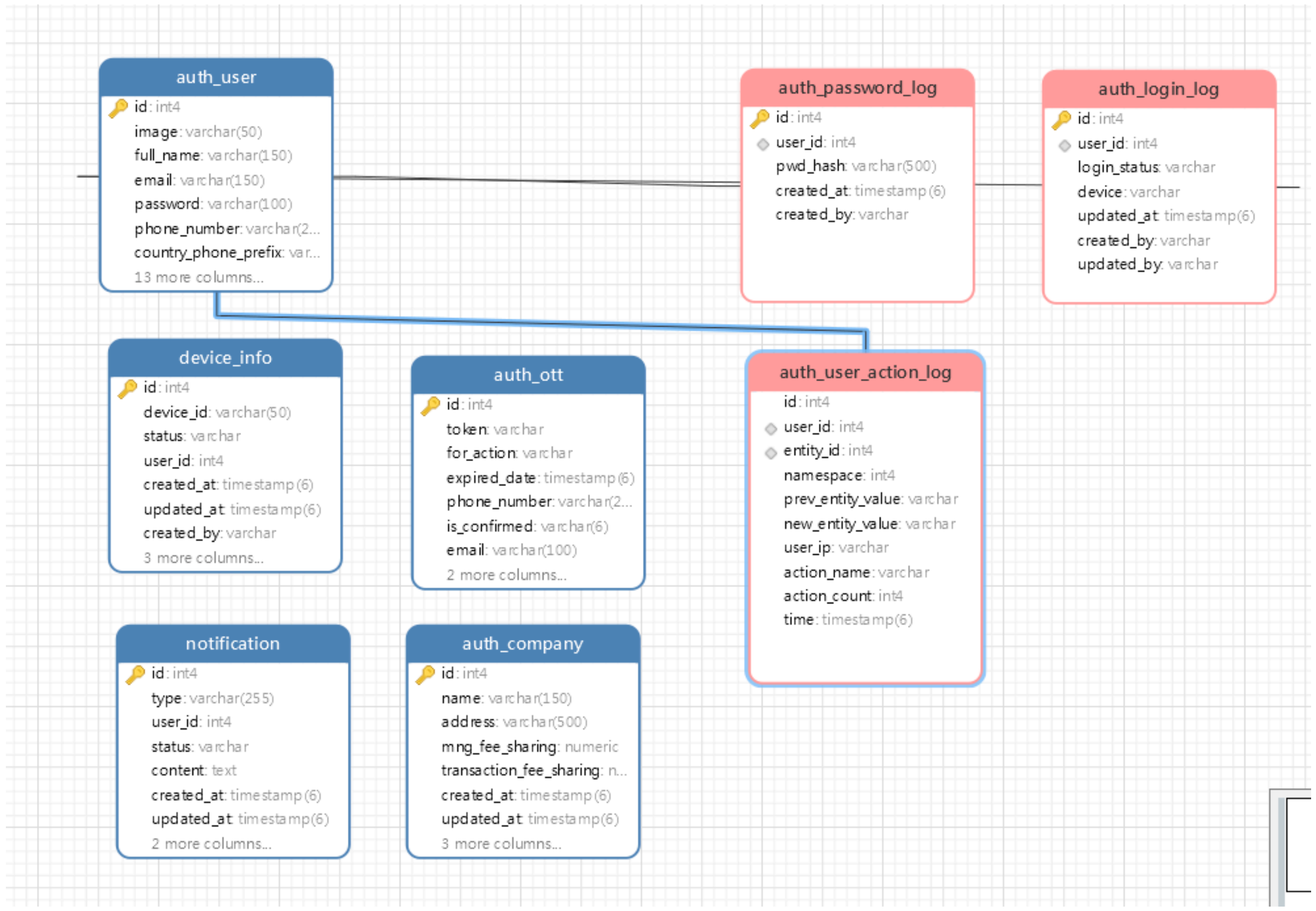
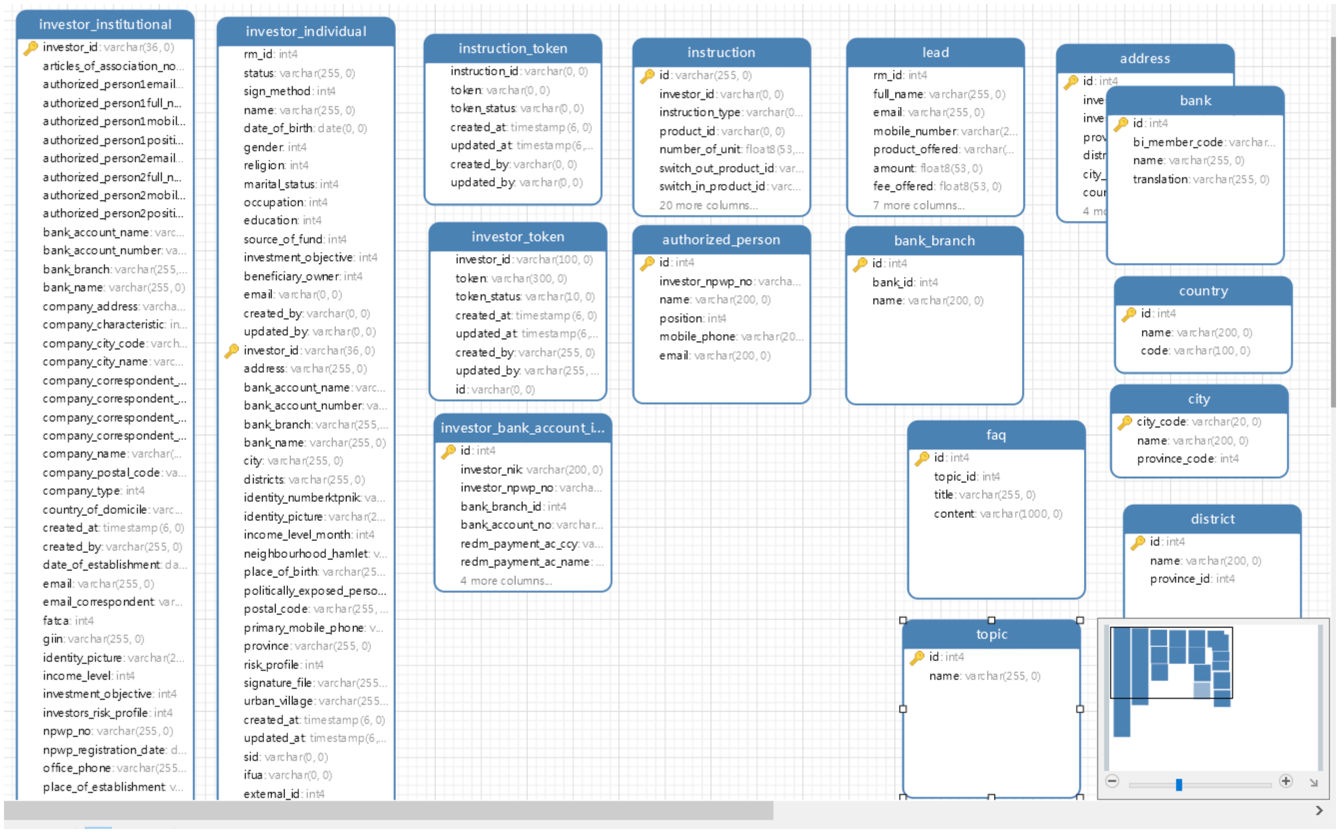


Table	Description:	Note
auth_company	Store the tenant/company that RM belong to	Primary key: ID (db auto-increment)
auth_ott	Store OTT/OTP	Primary key: ID (UUID v4) for avoid brutal force scanning.
auth_user	Store User	Primary key: ID (db auto-increment)
device_info	Store Devices	Primary key: ID (db auto-increment)
notification	Store notification	Primary key: ID (db auto-increment)
auth_login_log	Check login fail	Primary key: ID (db auto-increment)
auth_password_log	Check password change	Primary key: ID (db auto-increment)
auth_user_action_log	Audit trail	Primary key: ID (db auto-increment)

CRM µService



investor_institutional
investor_id: varchar(36, 0)
articles_of_association_no...
authorized_person1email...
authorized_person1full_n...
authorized_person1mobil...
authorized_person1positi...
authorized_person2email...
authorized_person2full_n...
authorized_person2mobil...
authorized_person2positi...
bank_account_name: varc...
bank_account_number: va...
bank_branch: varchar(255,...
bank_name: varchar(255, 0)
company_address: varcha...
company_characteristic: in...
company_city_code: varch...
company_city_name: varc...
company_correspondent_...
company_correspondent_...
company_correspondent_...
company_correspondent_...
company_name: varchar(...
company_postal_code: va...
company_type: int4
country_of_domicile: varc...
created_at: timestamp(6, 0)
created_by: varchar(255, 0)
date_of_establishment: da...
email: varchar(255, 0)
email_correspondent: var...
fatca: int4
giin: varchar(255, 0)
identity_picture: varchar(2...
income_level: int4
investment_objective: int4
investors_risk_profile: int4
npwp_no: varchar(255, 0)
npwp_registration_date: d...
office_phone: varchar(255...
place_of_establishment: v...

investor_individual
rm_id: int4
status: varchar(255, 0)
sign_method: int4
name: varchar(255, 0)
date_of_birth: date(0, 0)
gender: int4
religion: int4
marital_status: int4
occupation: int4
education: int4
source_of_fund: int4
investment_objective: int4
beneficiary_owner: int4
email: varchar(0, 0)
created_by: varchar(0, 0)
updated_by: varchar(0, 0)
investor_id: varchar(36, 0)
address: varchar(255, 0)
bank_account_name: varc...
bank_account_number: va...
bank_branch: varchar(255,...
bank_name: varchar(255, 0)
city: varchar(255, 0)
districts: varchar(255, 0)
identity_numberktpnik: va...
identity_picture: varchar(2...
income_level_month: int4
neighbourhood_hamlet: v...
place_of_birth: varchar(25...
politically_exposed_perso...
postal_code: varchar(255, ...
primary_mobile_phone: v...
province: varchar(255, 0)
risk_profile: int4
signature_file: varchar(255...
urban_village: varchar(255...
created_at: timestamp(6, 0)
updated_at: timestamp(6,...
sid: varchar(0, 0)
ifua: varchar(0, 0)
external_id: int4

instruction_token
instruction_id: varchar(0, 0)
token: varchar(0, 0)
token_status: varchar(0, 0)
created_at: timestamp(6, 0)
updated_at: timestamp(6,...
created_by: varchar(0, 0)
updated_by: varchar(0, 0)

investor_token
investor_id: varchar(100, 0)
token: varchar(300, 0)
token_status: varchar(10, 0)
created_at: timestamp(6, 0)
updated_at: timestamp(6,...
created_by: varchar(255, 0)
updated_by: varchar(255, ...
id: varchar(0, 0)

investor_bank_account_i...
id: int4
investor_nik: varchar(200, 0)
investor_npwp_no: varcha...
bank_branch_id: int4
bank_account_no: varchar...
redm_payment_ac_ccy: va...
redm_payment_ac_name: ...
4 more columns...

instruction
id: varchar(255, 0)
investor_id: varchar(0, 0)
instruction_type: varchar(0...
product_id: varchar(0, 0)
number_of_unit: float8(53,...
switch_out_product_id: var...
switch_in_product_id: varc...
20 more columns...

authorized_person
id: int4
investor_npwp_no: varcha...
name: varchar(200, 0)
position: int4
mobile_phone: varchar(20...
email: varchar(200, 0)

lead
rm_id: int4
full_name: varchar(255, 0)
email: varchar(255, 0)
mobile_number: varchar(2...
product_offered: varchar(...
amount: float8(53, 0)
fee_offered: float8(53, 0)
7 more columns...

bank_branch
id: int4
bank_id: int4
name: varchar(200, 0)

address
id: int4
inve
inve
prov
distr
city_
cour
4 mo

bank
id: int4
bi_member_code: varchar...
name: varchar(255, 0)
translation: varchar(255, 0)

country
id: int4
name: varchar(200, 0)
code: varchar(100, 0)

city
city_code: varchar(20, 0)
name: varchar(200, 0)
province_code: int4

faq
id: int4
topic_id: int4
title: varchar(255, 0)
content: varchar(1000, 0)

district
id: int4
name: varchar(200, 0)
province_id: int4

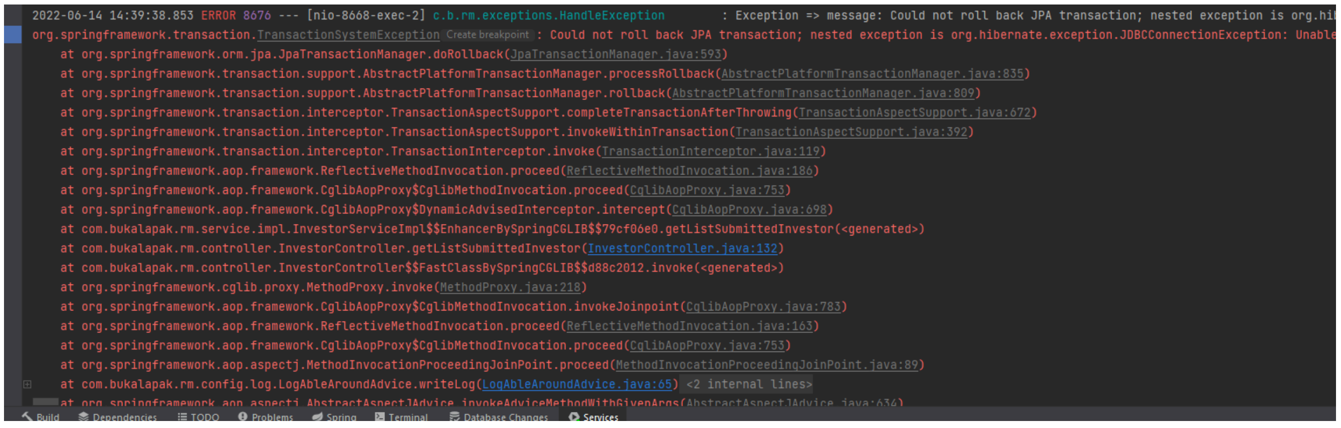
topic
id: int4
name: varchar(255, 0)

Table	Description:	Note
investor_individual	Store indie investor metadata.	Primary key: ID (UUID v4) for avoid brutal force scanning.
investor_institutional	Store the investor instie metadata.	Primary key: ID (UUID v4) for avoid brutal force scanning.
investor_token	Store token for some action need investor handle: sign, remote via email	Primary key: ID (db auto-increment)
investor_bank_account	Store the bank account of investor.	Primary key: ID (db auto-increment)
instruction	Store investor transaction	Primary key: ID (UUID v4) for avoid brutal force scanning.
instruction_token	Store token for some actions need investor handle: approve via email	Primary key: ID (db auto-increment)
lead	Store potential investor	Primary key: ID (db auto-increment)
bank	Store all bank in Indonesia	
country	Store country name in the world	
province	Store province name	
city	Store city name	
district	Store district	

<u>Logging details:</u>

Error logging (using log4j). We log only **log.error** and some **log.info** to std.out.

In local env, can check directly in the console (or can grep after in syslogd or journald for several days)

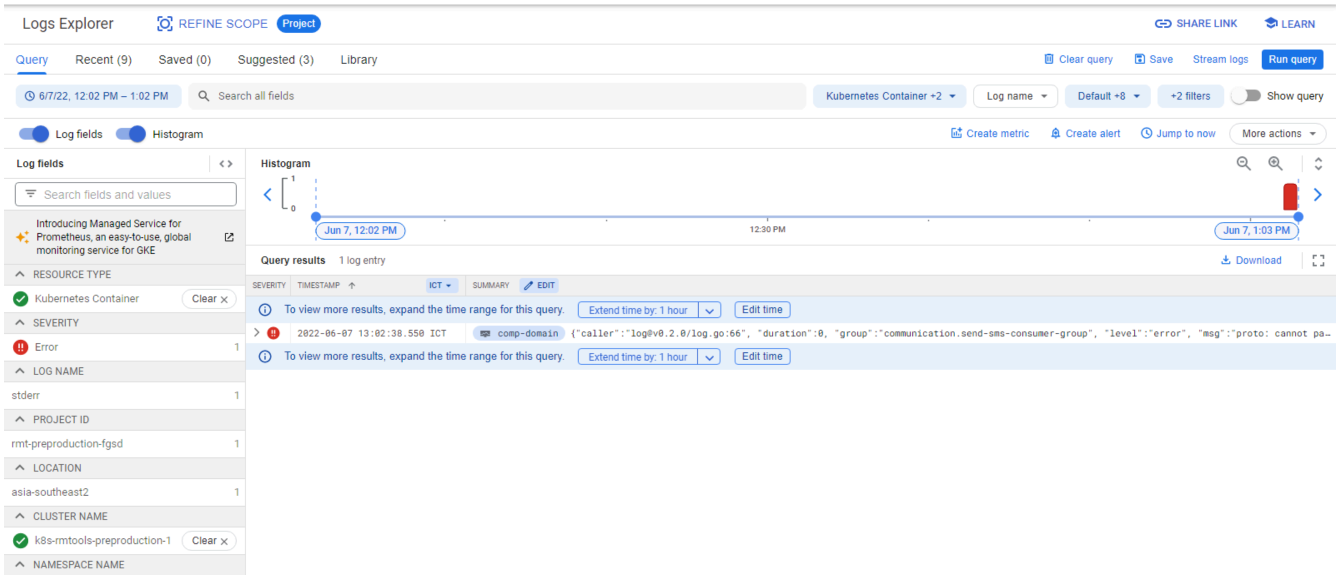


```
2022-06-14 14:39:38.853 ERROR 8676 --- [nio-8668-exec-2] c.b.rm.exceptions.HandleException        : Exception => message: Could not roll back JPA transaction; nested exception is org.hil
org.springframework.transaction.TransactionSystemException Create breakpoint : Could not roll back JPA transaction; nested exception is org.hibernate.exception.JDBCConnectionException: Unable
    at org.springframework.orm.jpa.JpaTransactionManager.doRollback(JpaTransactionManager.java:593)
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.processRollback(AbstractPlatformTransactionManager.java:835)
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.rollback(AbstractPlatformTransactionManager.java:809)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.completeTransactionAfterThrowing(TransactionAspectSupport.java:672)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:392)
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:119)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:186)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:753)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:698)
    at com.bukalapak.rm.service.impl.InvestorServiceImpl$$EnhancerBySpringCGLIB$$79cf06e0.getListSubmittedInvestor(<generated>)
    at com.bukalapak.rm.controller.InvestorController.getListSubmittedInvestor(InvestorController.java:132)
    at com.bukalapak.rm.controller.InvestorController$$FastClassBySpringCGLIB$$d88c2012.invoke(<generated>)
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:783)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:753)
    at org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedingJoinPoint.java:89)
    at com.bukalapak.rm.config.log.LogAbleAroundAdvice.writeLog(LogAbleAroundAdvice.java:65) <2 internal lines>
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethodWithGivenArgs(AbstractAspectJAdvice.java:634)
```

In preproduction, for real-time log, please use this command:

kubectl logs --tail=200 -f rmtools-apiauth-6765c85bc6-vtzfr -n rmtools rmtools-apiauth-6765c85bc6-vtzfr	Auth µService
kubectl logs --tail=200 -f rmtools-apiauth-6765c85bc6-vtzfr -n rmtools rmtools-apicrm-5ccc47d666-9tgvv	CRM µService
kubectl logs --tail=200 -f rmtools-apiauth-6765c85bc6-vtzfr -n rmtools rmtools-apibff-968ccbc4b-2x5tg	BFF

In preproduction, could use LogExplorer (UI) but will have delay ~ 30s.



6/7/22, 12:02 PM – 1:02 PM | Kubernetes Container +2 | Log name | Default +8 | +2 filters

Introducing Managed Service for Prometheus, an easy-to-use, global monitoring service for GKE

Histogram: Jun 7, 12:02 PM — 12:30 PM — Jun 7, 1:03 PM

RESOURCE TYPE	
Kubernetes Container	Clear ×
SEVERITY	
Error	1
LOG NAME	
stderr	1
PROJECT ID	
rmt-preproduction-fgsd	1
LOCATION	
asia-southeast2	1
CLUSTER NAME	
k8s-rmtools-preproduction-1	Clear ×
NAMESPACE NAME	

Query results 1 log entry

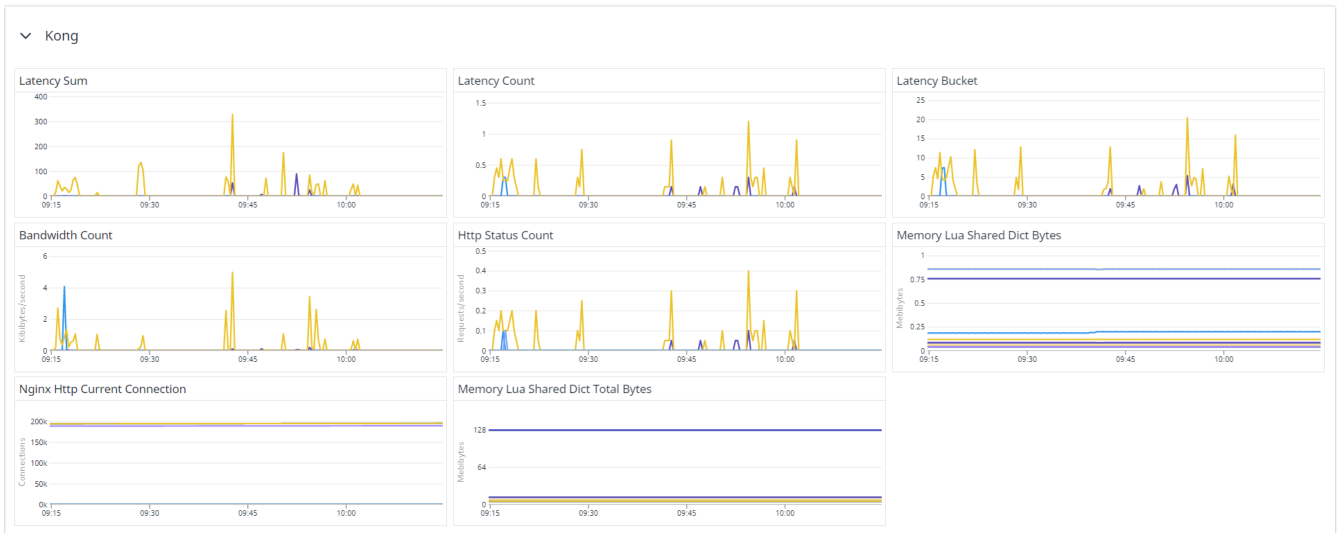
SEVERITY TIMESTAMP ICT SUMMARY

To view more results, expand the time range for this query.

```
2022-06-07 13:02:38.550 ICT  comp-domain {"caller":"log@v0.2.0/log.go:66", "duration":0, "group":"communication.send-sms-consumer-group", "level":"error", "msg":"proto: cannot pa…
```

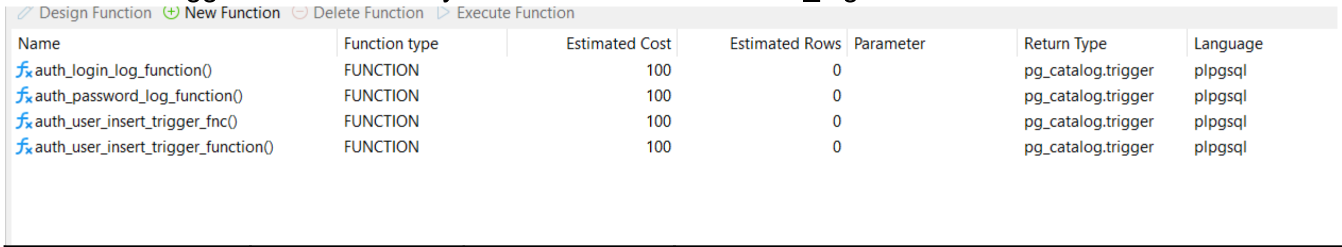
To view more results, expand the time range for this query.

In preproduction, please check Kong Dashboard in Datadog



Audit trails and trigger function

We use DBTrigger to automatically insert one record to action_log for audit trail.



Name	Function type	Estimated Cost	Estimated Rows	Parameter	Return Type	Language
auth_login_log_function()	FUNCTION	100	0		pg_catalog.trigger	plpgsql
auth_password_log_function()	FUNCTION	100	0		pg_catalog.trigger	plpgsql
auth_user_insert_trigger_fnc()	FUNCTION	100	0		pg_catalog.trigger	plpgsql
auth_user_insert_trigger_function()	FUNCTION	100	0		pg_catalog.trigger	plpgsql

Procedure	Trigger when	Description	Script

auth_login_log_function	App call API_login	Tracking login success or not.	CREATE OR REPLACE FUNCTION "public"."auth_login_log_function"()
			RETURNS "pg_catalog"."trigger" AS $BODY$
			BEGIN
			IF (NEW.last_login <> OLD.last_login) THEN
			INSERT INTO auth_bib.auth_login_log(user_id, device, updated_by, updated_at, created_by, login_status)
			VALUES (NEW.id, NEW.workspace, NEW.updated_by, now(), NEW.created_by, 'OK');
			END IF;
			IF (NEW.login_fail_count = OLD.login_fail_count + 1) THEN
			INSERT INTO auth_bib.auth_login_log(user_id, device, updated_by, updated_at, created_by, login_status)
			VALUES (NEW.id, NEW.workspace, NEW.updated_by, now(), NEW.created_by, 'FAIL');
			END IF;
			RETURN NULL;
			END;
			$BODY$
			LANGUAGE plpgsql VOLATILE
			COST 100

auth_password_log_function	App call API_change_password	Tracking password change.	CREATE OR REPLACE FUNCTION "public"."auth_password_log_function"()
			RETURNS "pg_catalog"."trigger" AS $BODY$
			BEGIN
			IF (NEW.password <> OLD.password) THEN
			if ((select count(*) as totalRecord from auth_bib.auth_password_log where user_id = NEW.id) < 5) then

			insert into auth_bib.auth_password_log(user_id, pwd_hash, created_at, created_by)
			values (NEW.id, NEW.password, now(), NEW.created_by);
			else
			delete
			from auth_bib.auth_password_log al
			where al.id in (select al.id
			from auth_bib.auth_password_log al
			where al.user_id = NEW.id
			order by al.created_at asc
			limit 1);
			end if;
			RETURN NULL;
			END IF;
			END;
			$BODY$
			LANGUAGE plpgsql VOLATILE
			COST 100

auth_user_insert_trigger_function	App call if any add in user, transaction, lead, company (params = entity)	Audit trail	CREATE OR REPLACE FUNCTION "public"."auth_user_insert_trigger_function"()
			RETURNS "pg_catalog"."trigger" AS $BODY$
			BEGIN
			INSERT INTO "auth_bib"."auth_user_action_log" ("entity", "entity_id", "namespace", "prev_entity_value", "new_entity_value", "user_ip", "action_name", "created_at", "created_by")
			VALUES ('user', NEW.id, NEW.workspace, '', row_to_json(NEW), '127.0.0.1', 'add', now(), NEW.created_by);
			RETURN NULL;
			END;
			$BODY$
			LANGUAGE plpgsql VOLATILE
			COST 100

auth_user_update_trigger_function	App call if any update in user, transaction, lead, company (params = entity)	Audit trail	CREATE OR REPLACE FUNCTION "public"."auth_user_update_trigger_function()"()
			RETURNS "pg_catalog"."trigger" AS $BODY$
			BEGIN
			INSERT INTO "auth_bib"."auth_user_action_log" ("entity", "entity_id", "namespace", "prev_entity_value", "new_entity_value", "user_ip", "action_name", "created_at", "created_by")
			VALUES ('user', OLD.id, NEW.workspace, row_to_json(OLD), row_to_json(NEW), '127.0.0.1', 'update', now(), NEW.updated_by);
			RETURN NULL;
			END;
			$BODY$
			LANGUAGE plpgsql VOLATILE
			COST 100