B2B SaaS開発

Configurabilityとマルチテナントをがんばる

α版編

A1A株式会社 @__mnc90

佐々木 延也

@__mnc90

A1A株式会社

サーバーサイドエンジニア

株式会社カカクコム

新卒で入社し食べログの予約システム全般と
Androidアプリの開発を担当

株式会社Speee

架電システムのSPA化を担当後、
SREとして事業部の開発基盤構築・運用を行う

A1A株式会社

創業メンバーとして参加
製造業の購買業務を効率化するSaaS RFQクラウド
を開発中

注意

本日のお話はα版のものなので
設計はまだ変わる可能性があります

AGENDA

1. SaaSとは
2. システムに求められること
3. RFQクラウドの説明
4. マルチテナントアーキテクチャ
5. Configurability
6. まとめ

1. SaaSとは

SaaSの定義

”SaaS（サース、Software as a Service）は、必要な機能を必要な分だけサービスとして利用できるようにしたソフトウェア（主にアプリケーションソフトウェア）もしくはその提供形態のこと。
一般には**インターネット経由**で必要な機能を利用する仕組みで、**シングルシステム・マルチテナント方式**になっているものを指す”

引用: https://ja.wikipedia.org/wiki/SaaS

法人向けシステムの変遷

①スクラッチ開発

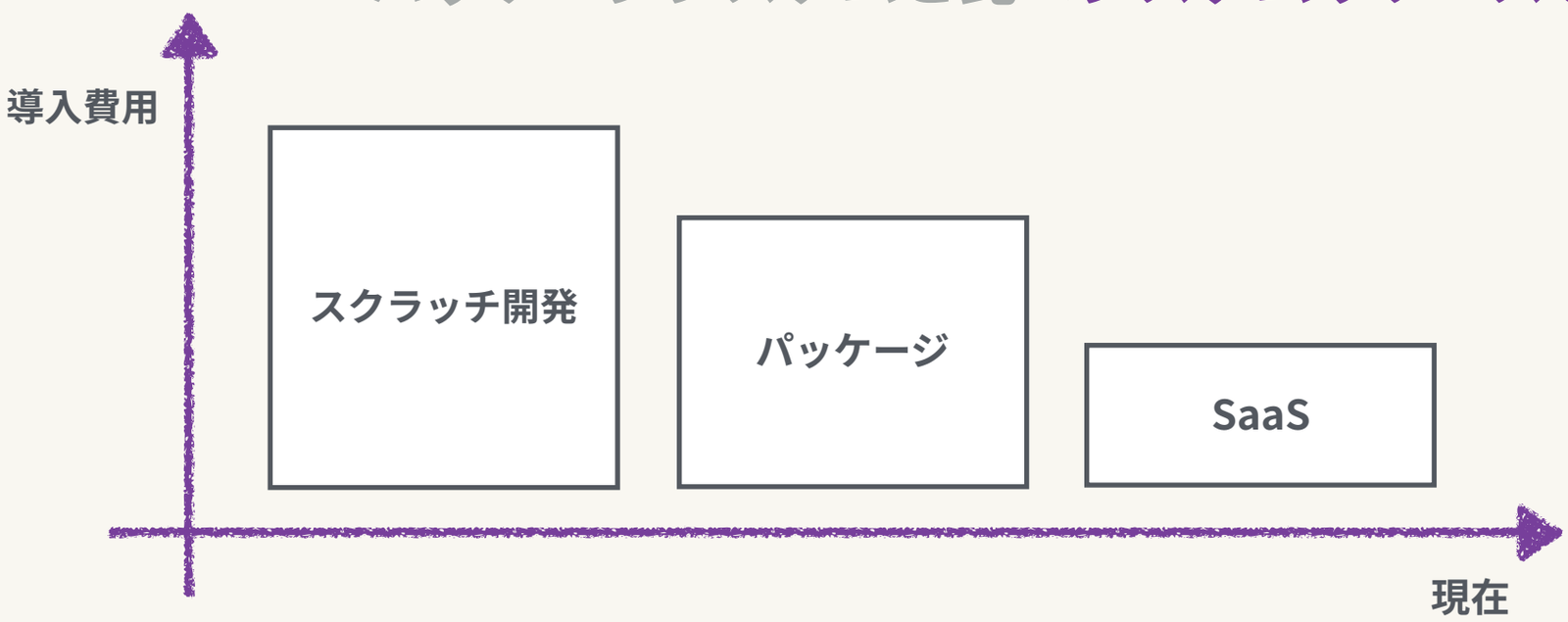
企業毎に開発をしていたため初期導入費用・運用費用が高額

②パッケージ

パッケージ化したことで初期導入費用を抑えた

③SaaS

パッケージシステムと比べ**システムリソースを共有化**し低価格化



SaaSのメリット

①低価格

クラウドインフラによりシステムリソースを共有できるため、低価格で提供できる

②初期導入工数・導入費用が少ない

スクラッチ開発やパッケージと比べ、インフラ構築が不要なため導入工数が少ない

③スモールスタートがしやすい

サブスクリプションモデルで初期費用が安いため、気軽にはじめることができる

④継続的にシステムに機能追加・改善が行われる

汎用的な機能が追加されていくため新機能を追加費用なく利用できる

SaaSのデメリット

①企業毎の要望に対応しづらい

企業毎の個別開発を行うと低価格のサブスクリプションというビジネスモデルだと採算がとれなくなるため、原則SaaSベンダーは個別の開発対応を行わない。

スクラッチ開発は言わずもがな、パッケージシステムでもカスタマイズ開発が可能なものが多いため、SaaSのデメリットといえる。

2. システムに求められること

SaaSのメリットを伸ばす

保守開発の工数を少なくしシステムの利用効率を高め低価格化を実現

⇛これを実現できるような**マルチテナントアーキテクチャ**で実装する必要がある。

SaaSのデメリットを軽減する

Configurabilityを確保したプロダクト開発

企業毎に異なる多様な要望に対して都度のシステム開発を行わずに対応するため、設定で機能をカスタマイズできるようにする。

企業間のデータが混じらないように分離された設計

オンプレと比べ、**「クラウド x マルチテナント」**はセキュリティに関する懸念を持たれることが多い。

そのため、あらゆるデータソースについてバグなどで企業間のデータが混じらないような安全性の高い設計が求められる。

まとめると

システムには以下が求められる

- システム利用効率が高く安全性の高いマルチテナントアーキテクチャ
- Configurabilityを確保したプロダクトの開発
- セキュリティ・インシデント対策

2. システムに求められること

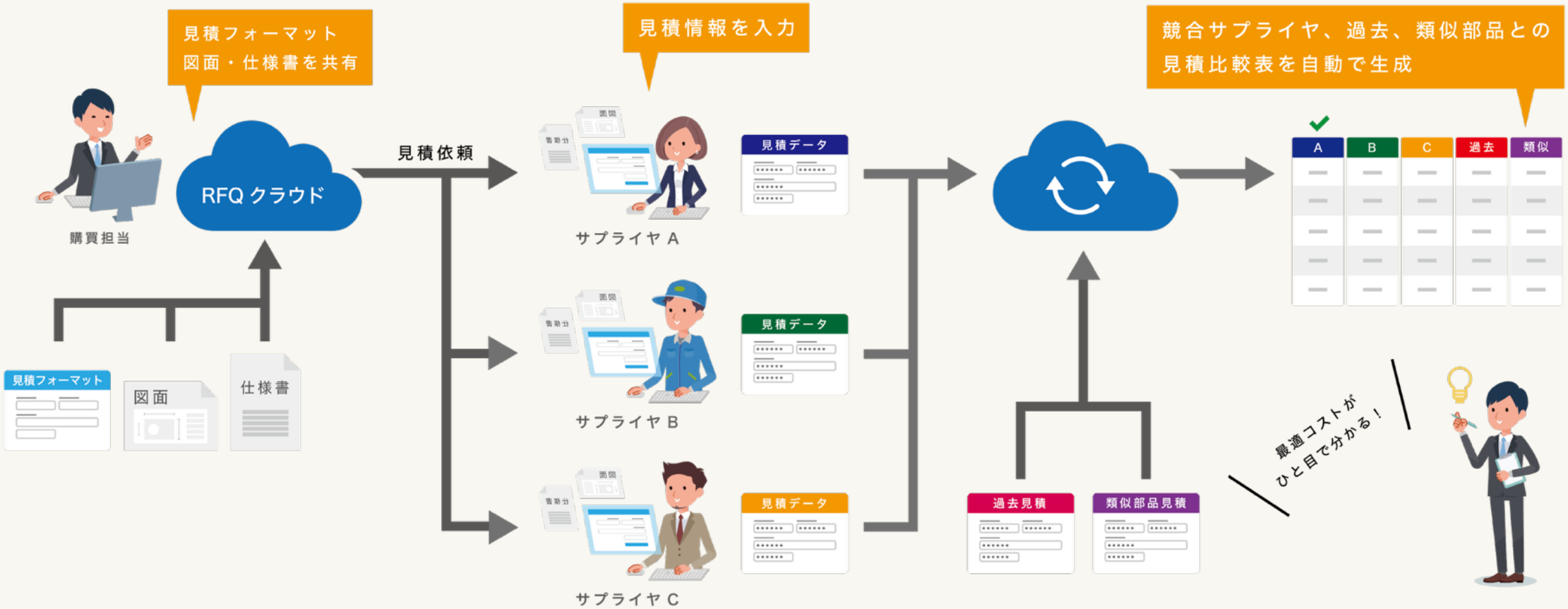
これらをRFQクラウドではどう実現しているか

3. RFQクラウドの説明

3. RFQクラウドの説明

RFQクラウドとは

製造業の調達・購買担当者の見積取得プロセス(RFQ)を最適化し
コスト削減を支援するSaaS



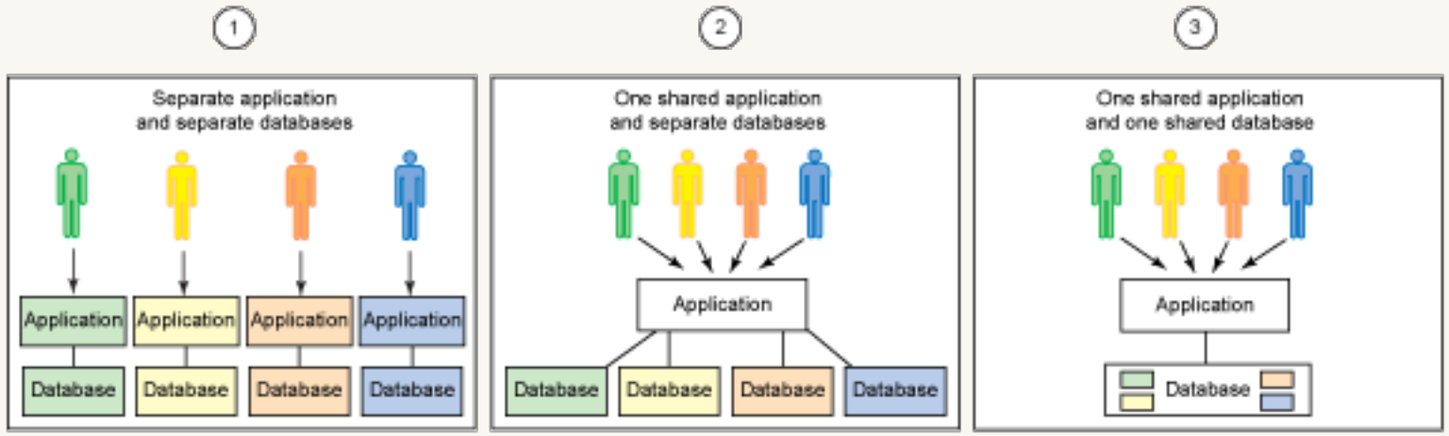
4. マルチテナントアーキテクチャ

マルチテナントアーキテクチャとは

”マルチテナント方式*1とは、**1つのシステム**の中に**複数の企業**（ユーザー）のサービスを同居させ、リソースや運用コストを大幅に低減する方式のことである。”

引用: http://www.atmarkit.co.jp/fdotnet/bookpreview/azureoverview_0301/azureoverview_0301_01.html

マルチテナンシーのレベル



①クラウド内での単純な仮想化によりハードウェアのみを共有

②1アプリケーションでテナントごとに異なるデータベースを使用

③1アプリケーションでデータベースを共有

引用: https://www.ibm.com/developerworks/jp/cloud/library/cl-multitenantsaas/

スキーマ共有方式の事例①

共有データベースサーバーでテナント毎にスキーマを分離

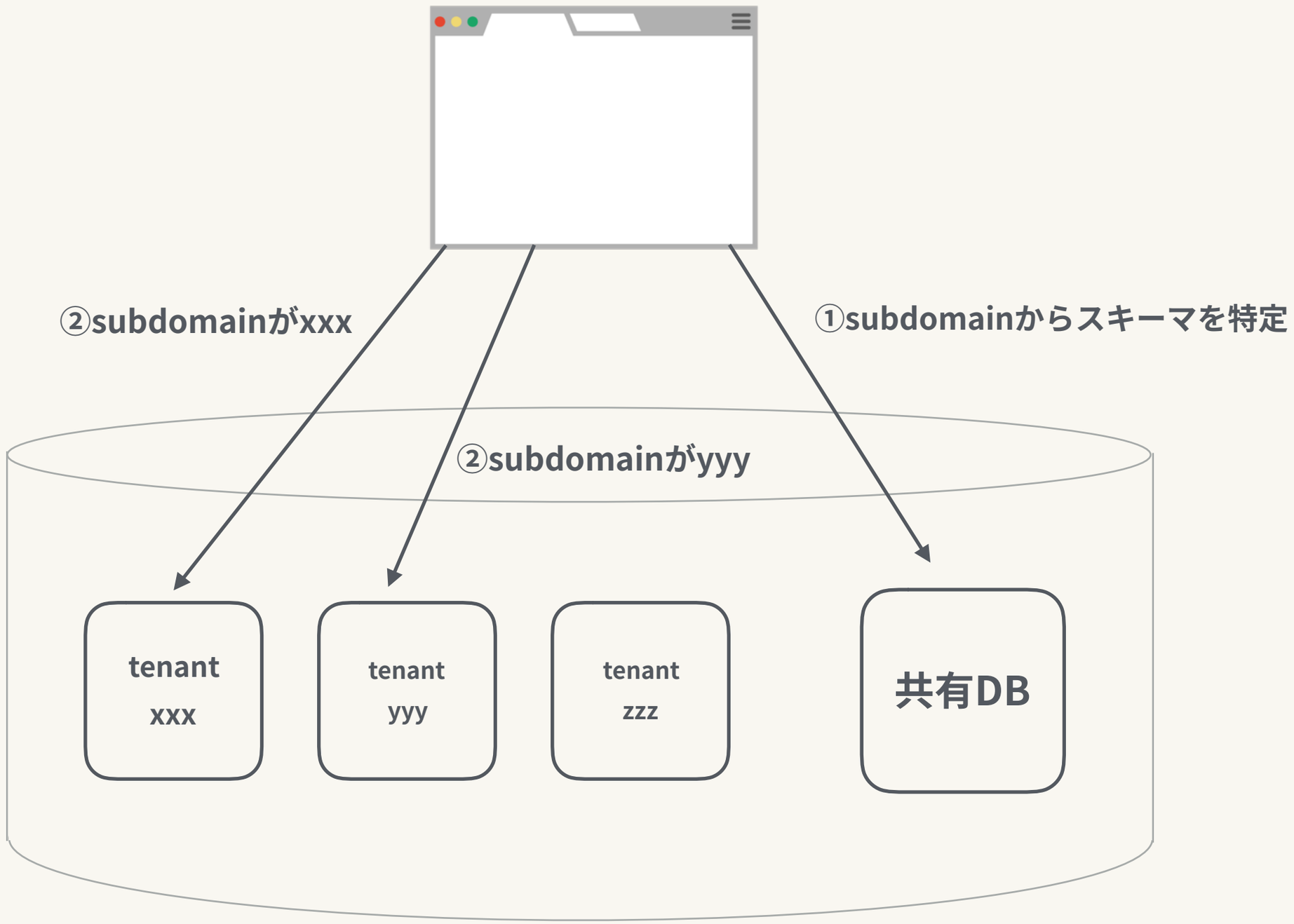
PostgreSQLならSchema、MySQLならDatabaseを分離する。

SmartHR, Kibela(BitJourney)が採用。

SmartHRは次ページの方式に変更したそう。

Rubyでは **influitive/apartment** がある

スキーマ共有方式の事例①



スキーマ共有方式の事例②

全テーブルにtenant_idカラムを用意する

SalesforceやSmartHRが採用。

Rubyでは **citusdata/activerecord-multi-tenant** や **ErwinM/acts_as_tenant** がある

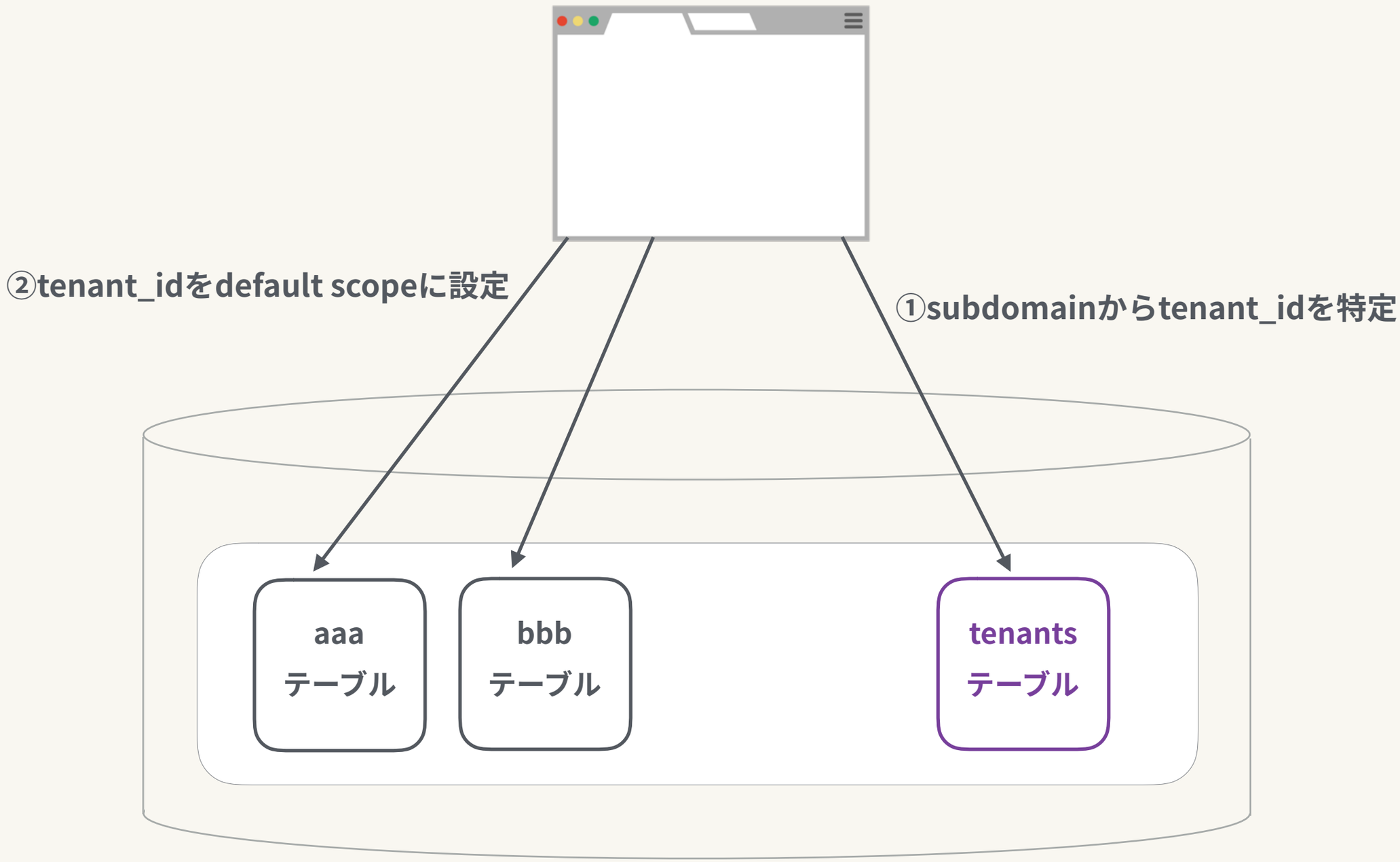
users

id	tenant_id	name	email
1	1	佐々木	sasaki@a1a.co.jp
2	3	松本	matsumuto@a1a.co.jp

materials

id	tenant_id	name
1	1	フロントパネル
2	3	ファンヒーター

スキーマ共有方式の事例②



RFQクラウドでは②を採用

①migrationの時間がO(n)で増加。

※SmartHRやKibelaの知見を参考にしました🙏

SmartHRではmigrationに数時間かかっていた。

RFQクラウドでは将来的に企業間取引のネットワークを作りたいため、テナント数は万単位になることが想定される。

②DDLのRollbackができない

※PostgreSQLはできるようだが

そのため、スケールしやすい②の方式を採用することにした。

4. マルチテナントアーキテクチャ

しかしいくつもの課題が…

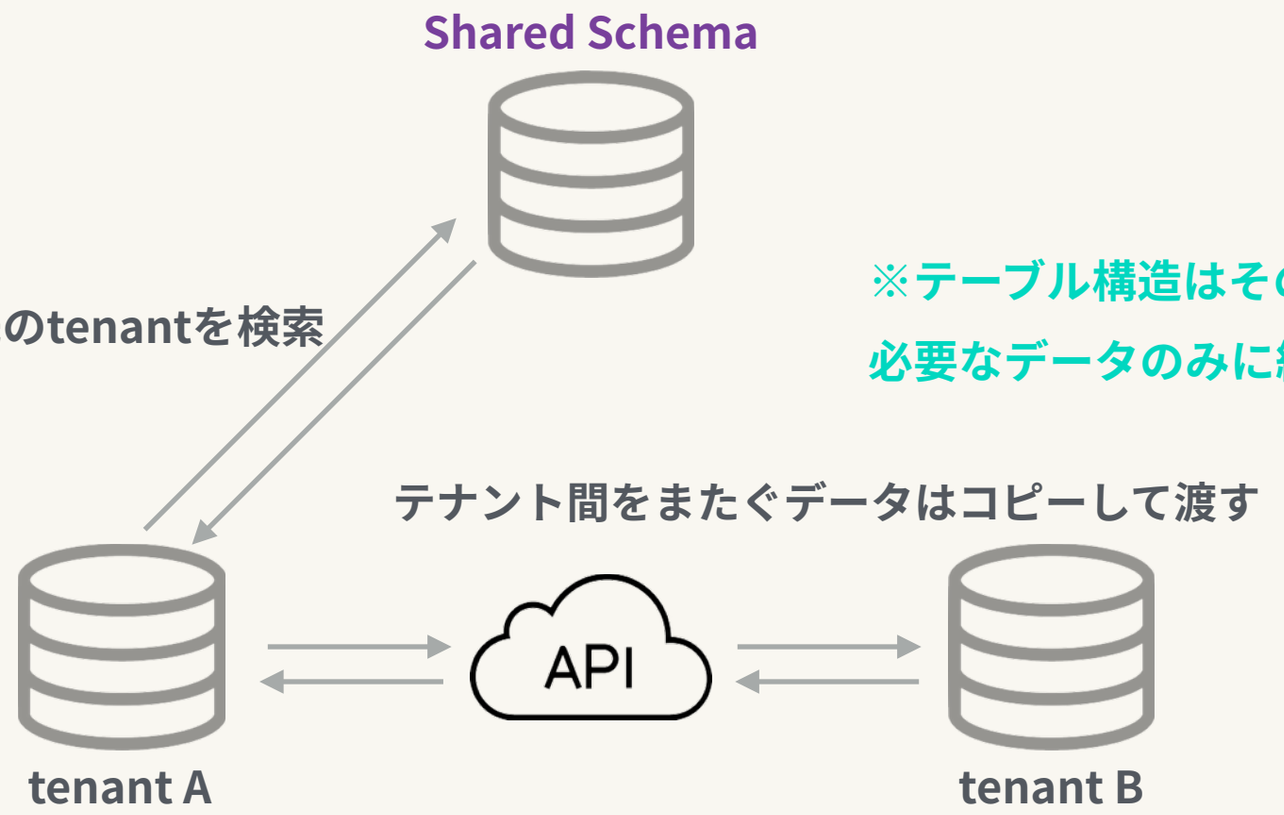
テナント間コミュニケーション

見積依頼とその回答ではテナント間でデータのやり取りが発生する

マルチテナントアーキテクチャを実装する場合、別テナントのデータを直接見ることは避けたい

しかし共有スキーマを用意する場合、別テナントのデータが見れてしまうバグなどをうむ可能性がある

テナント間コミュニケーション



※テーブル構造はそのまま再現せず
必要なデータのみに絞って非正規化して渡している

海外のSaaS Tradeshiftを参考にテナント間でやり取りするデータはコピーしてAPIで渡す設計に
そうすることでテナント外のデータについては意識する必要がなくなりテナント毎の隔離性を損なわなくなる

設計のポイント

①マスターデータとトランザクションデータでコピーのタイミングが異なる

マスターデータは取引先として登録したタイミングでコピーする必要がある。でないとトランザクション時(見積依頼時)のデータ量が多くなる。

②マスターデータが更新された場合にコピー先のデータも更新する必要がある

トランザクションデータはビジネス要件で更新は禁止だが、マスターデータは更新する必要がある。(例) 見積回答時の選択肢のマスターデータ

設計のポイント

③コピー元のテーブル構造をそのまま表現しようせず非正規化する

メールと同じように一度送信した内容は更新できないという整理をした

そのため送信するデータを非正規化して保持するようにした

④取引先のレコードは直接見れるがあえてJob経由でやり取りする

アプリケーション開発者はテナント企業を意識せずに実装したい。

そのため、別テナントとの送信/受信を明示的に非同期でわける(*)ことでテナントを意識させないような設計にした

※引数にtenant_idを渡して取引先テナントにデータを作成するといったやり方にはしない

設計のポイント

⑤取引先アカウントの招待時にはサプライヤー用のテナントを作成する

招待されたアカウントをバイヤーテナント配下には作成せずに、そのアカウントの企業のテナントを作成し、その中のアカウントとして作成するようにした。

こうすることで以下を実現した

・サプライヤー側に(招待される側)複数の重複アカウントが作られることを防ぐ

・サプライヤーの回答データという資産をサプライヤーが整理して保持できる

また、企業合併時と企業分裂時の辛さを比較して分裂の方がマシだと判断し、1企業1テナントとした。

企業毎のパスワードポリシー

Cognitoでは作成できるUser Pool数に上限があった

企業ごとに求めるセキュリティレベルが異なるため、企業毎にパスワードポリシーを設定できる必要があった

しかしCognitoでは作成できるUser Pool数の上限があったため、テナント毎にUser Poolを用意することはできない。

Auth0でテナント毎にconnection(User Poolに相当)を作成する

テナント作成時にそのテナント用のconnectionを動的に作成している。

※パスワードポリシー…半角英数字15文字以上など

5. Configurability

ConfigurabilityとCustomizabilityの違い

Configurability

顧客の多様な要望に対応するために行うことができる設定の変更可能性

顧客の要望を一般化し設定によって対応できるように開発することでベンダー側の導入コストを抑える

Customizability

顧客の要望に対応するために独自で開発を行う場合のカスタマイズ提供能力

顧客の要望が特殊であっても対応可能だが大きな導入コストが常に発生する

5. Configurability

CustomizabilityよりConfigurability

Configurabilityの難所、スキーマ不定データの扱い

スキーマ不定

スキーマは固定化できないが、その時点でのスキーマに適合しないデータは許容しない

スキーマレス

どんなデータの入力も受け付ける

(例) ログなど

5. Configurability

RFQクラウドのスキーマ不定データ

見積回答履歴 *

2018年11月25日 19:00

見積金額[円]	0
工程数	2

	工程名	新規改造区分	材料費[円]	購入部品費[円]	熱処理・メッキ等[円]	加工費[円]	設計費[円]	その他[円]	工程別見積金額[円]
1	123	新規製作	0	0	0	0	0	0	0
2	123	新規製作	0	0	0	0	0	0	0

備考

test

添付ファイル

スクリーンショット_2018-11-23_14.46.12.png

5. Configurability

RFQクラウドのスキーマ不定データ

見積回答履歴 *

2018年11月25日 19:00

見積金額[円]	0
工程数	2

	工程名	新規改造区分	材料費[円]	購入部品費[円]	熱処理・[illegible]円]	加工費[円]	設計費[円]	その他[円]	工程別見積金額[円]
1	123	新規製作	0	0	0	0	0	0	0
2	123	新規製作	0	0	0	0	0	0	0

項目名

備考

test

添付ファイル

スクリーンショット_2018-11-23_14.46.12.png

RFQクラウドのスキーマ不定データ

見積回答履歴 *

2018年11月25日 19:00

見積金額[円]	0
工程数	2

	工程名	新規改造区分	材料費[円]	購入部品費[円]	熱処理・メッキ等[円]	加工費[円]	設計費[円]	その他[円]	工程別見積金額[円]
1	123	新規製作	0	0	0	0	0	0	0
2	123	新規製作	0	0	0	0	0	0	0

整数

備考

test

添付ファイル

スクリーンショット_2018-11-23_14.46.12.png

RFQクラウドのスキーマ不定データ

見積回答履歴 *

2018年11月25日 19:00

見積金額[円]	0
工程数	2

	工程名	新規改造区分	材料費[円]	購入部品費[円]	熱処理・メッキ等[円]	加工費[円]	設計費[円]	その他[円]	工程別見積金額[円]
1	123	選択式	0	0	0	0	0	0	0
2	123	新規製作	0	0	0	0	0	0	0

備考

test

添付ファイル

スクリーンショット_2018-11-23_14.46.12.png

RFQクラウドのスキーマ不定データ

見積回答履歴 *

2018年11月25日 19:00

見積金額[円]	0
工程数	2

工程名	新規改造区分	材料費[円]	購入部品費[円]	熱処理・メッキ等[円]	加工費[円]	設計費[円]	その他[円]	工程別見積金額[円]
1	123	新規製作	0	0	0	0	0	0	0
2	123	新規製作	0	0	0	0	0	0	0

レコード数可変

備考

test

添付ファイル

スクリーンショット_2018-11-23_14.46.12.png

5. Configurability

スキーマ不定データのアプローチ

構造写像アプローチ

データの種別毎にRDBMSのテーブルを作成する

RailsならSingle table inheritance(単一テーブル継承)やClass table inheritance(クラステーブル継承)を使ってシンプルに実装可能

データの種別が大きく増えないのであれば使用可能

企業A用の見積テーブル

id	name	amount	price
1	SS10	100	1430
2	SS10A	100	920

企業B用の見積テーブル

id	name	amount	price	process
1	MNC	30	3220	熱処理
2	UNC20	50	4990	プレス

引用: http://dev.ariel-networks.com/Members/inoue/schemaless/

モデル写像アプローチ

カラム名と値をセットで持つレコードを作成する

SQLアンチパターンのEAV(Entity Attribute Value)に該当する

レコード数が増大する

見積テーブル

id	column	value	quote_id
1	工程	プレス	1
2	投入材料	鉄	1
3	投入量	100	1
4	工程	レーザー	2

引用: http://dev.ariel-networks.com/Members/inoue/schemaless/

Salesforceアプローチ

あらゆるデータを受け入れるDataテーブルを作成し、インデックス用テーブルなどのメタデータテーブルを用意する

Configurabilityの究極系。弊社ではOverkill感。

Salesforceの例

GUID	OrgID	ObjID	Value0	Value1	Value2	Value3	Value4	Value5
1	100	30	C-1	ホチキス	2009/2/8	250		
2	100	51	1	海山商事	田中	03-xxx-xxxx	aaa@umiyam	山城
3	100	51	2	四菱物産	武藤	03-yyy-yyyy	bbb@yotsubi	内藤
4	100	30	C-2	はさみ	2008/9/2	180		
5	100	30	C-3	定規	2008/6/1	150		
6	100	51	3	東西新聞	山岡	03-zzz-zzzz	ccc@touzai.c	海原
7	100	51	4	平成製菓	友田	-www-wwww	ddd@heisei.c	平沢

引用: https://www.publickey1.jp/blog/09/3_2.html

friendfeedアプローチ

Salesforceアプローチと似ているが、Dataテーブルを用意する代わりにJSONやXMLを入れるカラムを用意する

Salesforceアプローチ同様、Overkill感がする。

引用: http://dev.ariel-networks.com/Members/inoue/schemaless/

RFQクラウドのアプローチ

friendfeedアプローチ + ElasticSearch を選択

friendfeedアプローチのインデックステーブルは作成せず、検索はElasticSearchに任せる予定

データはJSONB型のカラムに保存

JSON型の場合はデータ更新が行いにくいというデメリットがあるが、システムの要件上データの変更は認めておらず、変更時はすべてレコードを新規追加するため無視できた

RFQクラウドのアプローチ

mozilla-services/react-jsonschema-formを利用

フォームのフォーマットやその入力データの表示などに利用

JSON Schemaをもとにフォームを生成するライブラリ

5. Configurability

RFQクラウドのアプローチ

見積テーブル

id	form_data

1	{
	"price": 595,
	"name": "MNC101"
	}

フォーマットテーブル

id	format

1	{
	"type": "object",
	"required": [
	"price"
	],
	"properties": {
	"price": {
	"type": "integer",
	"title": "費用"
	}
	}

mozilla-services/react-jsonschema-formを使ったテーブル構造

react-jsonschema-form

https://mozilla-services.github.io/react-jsonschema-form/

6. まとめ

6. まとめ

①マルチテナントアーキテクチャ下においてテナント間コミュニケーションが発生する場合は設計注意

②Configurabilityを担保するためにreact-jsonschema-formが使える

ββ版完成後にこの設計の良し悪しについて
また発表させてください🙏

Thanks!

A1A株式会社 @__mnc90