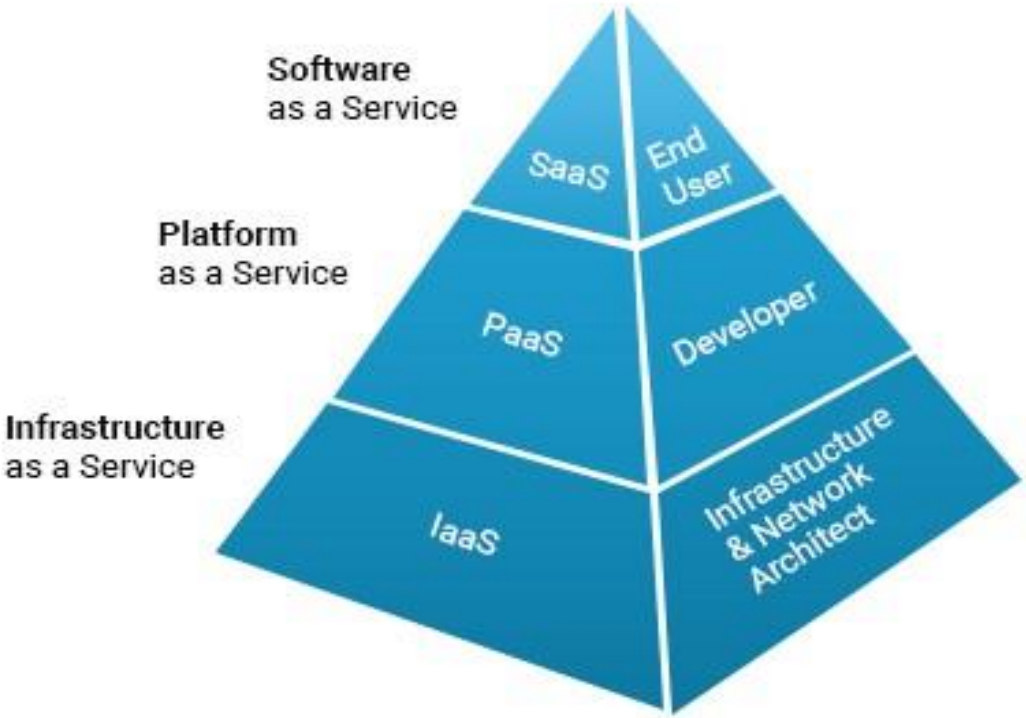
Building SaaS Applications with Django - A Multitenant Architecture

Timothy Olaleke
@timtech4u

Managing Multiple Instances can be a Pain...

What is Multi-tenancy

Multi-tenancy is an architectural approach which enables a single instance of an application to be shared among multiple organizations or users, which are also known as tenants.



Cloud Computing Models

Software as a Service (SaaS)

In building SaaS applications a multi-tenant architecture can be the best or only possible way out.

Factor to Consider in Building a SaaS Apps

- Multiple Clients

Factor to Consider in Building a SaaS Apps

- Multiple Clients
- New Releases

Factor to Consider in Building a SaaS Apps

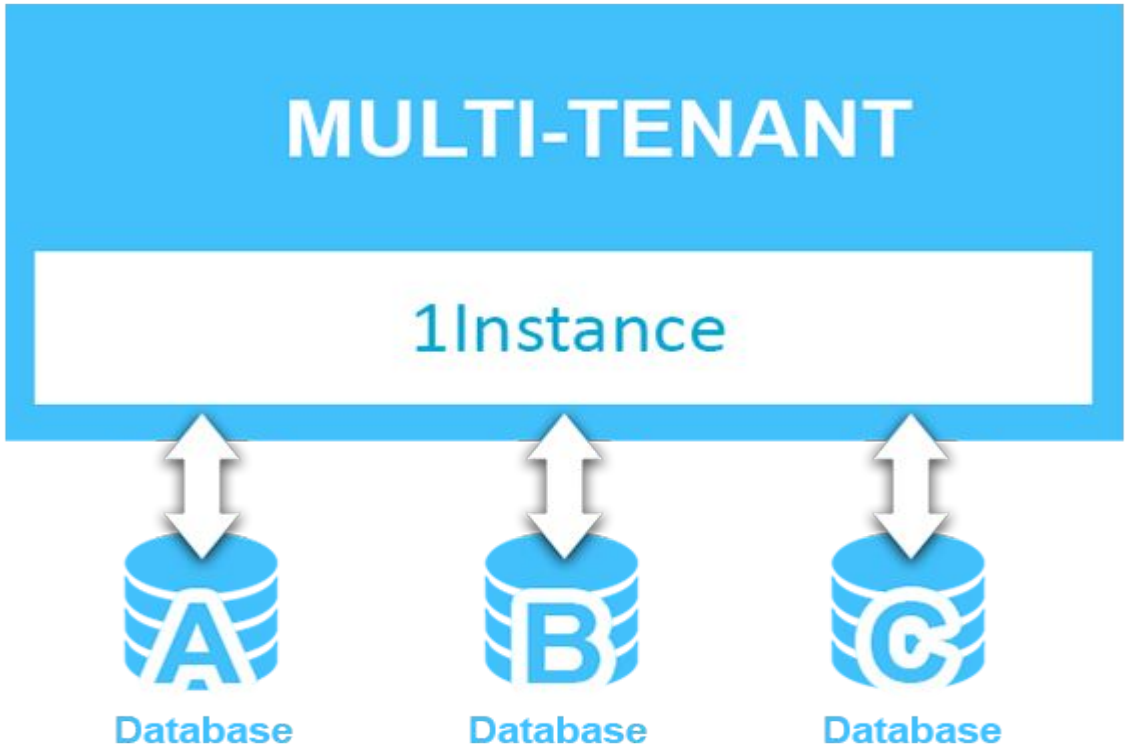
- Multiple Clients
- New Releases
- Deployment Costs & Time

Factor to Consider in Building a SaaS Apps

- Multiple Clients
- New Releases
- Deployment Costs & Time
- Scalability

Factor to Consider in Building a SaaS Apps

- Multiple Clients
- New Releases
- Deployment Costs & Time
- Scalability
- Single Codebase (Maintainability)

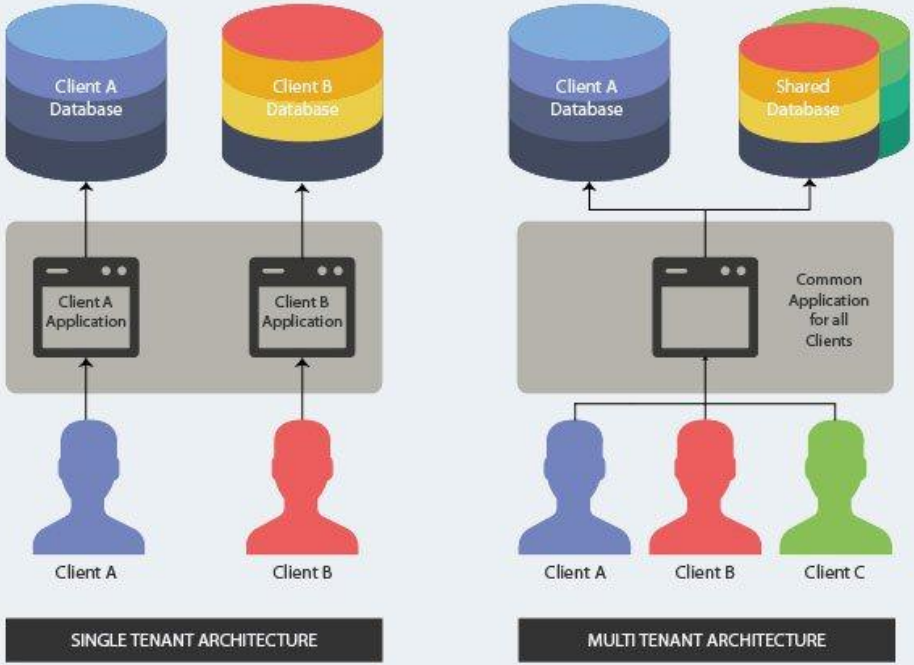


What it really looks like.

Popular Examples of SaaS Applications

- Slack
- G Suite
- Salesforce
- Shopify
- Hubspot CRM
- FlexiSAF

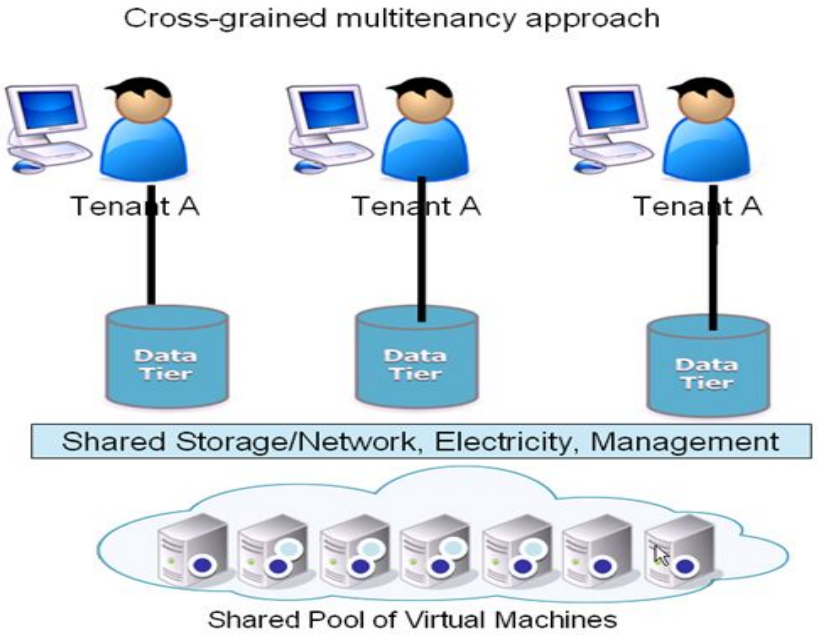
Multi-Tenant Architecture



Multi-tenancy are also grouped into Models.

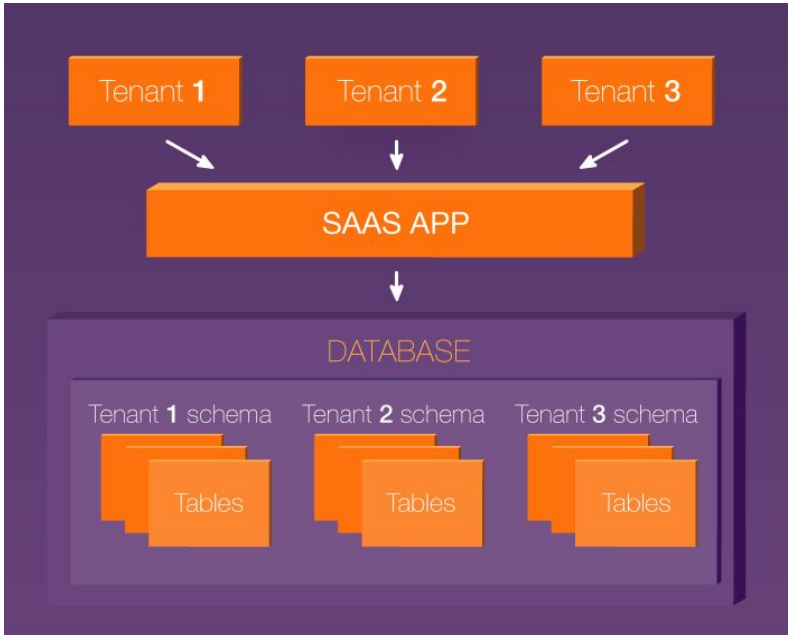
Common Multitenant Strategies

- **Isolated Approach** - Seperate Databases



Common Multitenant Strategies

- **Isolated Approach** -> Seperate Databases
- **Semi - isolated Approach** -> Multiple Databases



PostreSQL provides a single database with multiple schema database (each with multiple tables).

Multitenancy in Python(Django)

pip install django-tenant-schemas

django-tenant-schemas is based on the semi-isolated model which uses one database but multiple schemas.

This approach provides better security, simplicity and performance.


string sInput;
int iLength, iN;
double dblTemp;
bool again = true;
while (again) {
iN = -1;
again = false;
getline(cin, sInput);
system("cls");
stringstream(sInput) >> dblTemp;
iLength = sInput.length();
if (iLength < 4) {
again = true;
continue;
} else if (sInput[iLength - 3] != '.') {
again = true;
continue;
} while (++iN < iLength) {
if (isdigit(sInput[iN])) {
continue;
} else if (iN == (iLength - 3)) {

Tenant Model

```
from django.db import models
from tenant_schemas.models import TenantMixin

class Client(TenantMixin):
    name = models.CharField(max_length=100)
    paid_until =  models.DateField()
    on_trial = models.BooleanField()
    created_on = models.DateField(auto_now_add=True)
```

```
TENANT_MODEL = "customers.Client"
```

Shared and Tenant-Specific Apps

```
SHARED_APPS = (
    'tenant_schemas',  # mandatory, should always be before any django app
    'customers', # you must list the app where your tenant model resides in
    'django.contrib.contenttypes',
    'django.contrib.auth',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.admin',
)
TENANT_APPS = (
    'django.contrib.contenttypes',
     'django.contrib.auth',
     'myapp',
 )
 INSTALLED_APPS = TUPLE(SET(SHARED_APPS + TENANT_APPS))
```

Set Database Engine

```
DATABASES = {
    'default': {
        'ENGINE': 'tenant_schemas.postgresql_backend',
        # ...
    }
}
```

Set Database Routers

```
DATABASE_ROUTERS = (
    'tenant_schemas.routers.TenantSyncRouter',
)
```

Configure Middleware Classes & Template Context Processors

```
MIDDLEWARE_CLASSES = (
    'tenant_schemas.middleware.TenantMiddleware',
    # ...
)
```

```
TEMPLATE_CONTEXT_PROCESSORS = (
    'django.core.context_processors.request',
    #...
)
```

Commands

- `python manage.py makemigrations`
- `python manage.py migrate_schemas --shared`
- `python manage.py tenant_command createsuperuser --schema=eha-meetup`

Creating our first tenant - public

```
from customers.models import Client

tenant = Client(domain_url='mydomain.com',
  schema_name='public',
  name='Default',
  paid_until='2099-12-31',
  on_trial=False)
tenant.save()
```

Creating another new tenant - eha

```
from customers.models import Client

tenant = Client(domain_url='eha.mydomain.com',
  schema_name='eha',
  name='eha',
  paid_until='2099-12-31',
  on_trial=False)
tenant.save()
```

Still not convinced?

Benefits of SaaS Application to its Providers

- Consistent Revenue Stream

Benefits of SaaS Application to its Providers

- Consistent Revenue Stream
- Faster development cycles

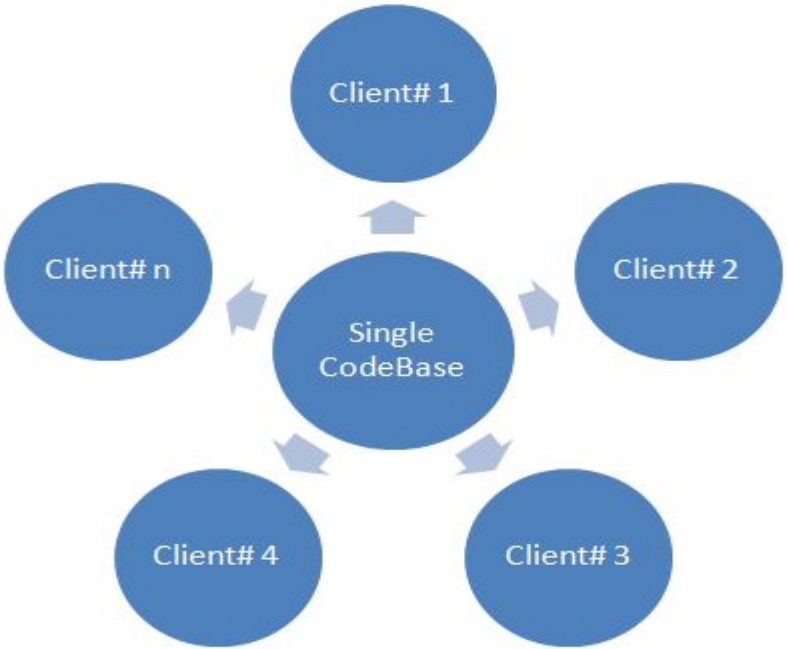
Benefits of SaaS Application to its Providers

- Consistent revenue stream.
- Faster development cycles.
- Easier troubleshooting of bugs.

Benefits of SaaS Application to its Providers

- Consistent revenue stream.
- Faster development cycles.
- Easier troubleshooting of bugs.
- Reduce risk of Software Piracy.

Typical SaaS CodeBase



Benefits of SaaS Application to its Users.

- Cheaper than running self hosted applications.

Benefits of SaaS Application to its Users.

- Cheaper than running self hosted applications.
- Reduced need for and cost associated with having an in-house IT team.

Benefits of SaaS Application to its Users.

- Cheaper than running self hosted applications.
- Reduced need for and cost associated with having an in-house IT team.
- Easily scalable up or down based on business needs.

Other Tips

- Use Wildcards e.g. *.mydomain.com

Other Tips

- Use Wildcards e.g. ***.mydomain.com**
- Consume API with JS Hostname
  http://${location.hostname}/api/

Other Tips

- Use Wildcards e.g. ***.mydomain.com**
- Consume API with JS Hostname

  http://${location.hostname}/api/
- Generate Wildcard(*) SSL certificates.

The Multitenant Architecture can also be adapted in other programming languages.

What other questions do you have for me???

Slides:

http://bit.ly/django-multitenant

Demo App:

https://github.com/Timtech4u/employee-manager

Django Tenant Schemas Docs:

https://django-tenant-schemas.readthedocs.io

Thanks for listening...