**《发布系统项目文档》**

|  |  |
| --- | --- |
| 作者信息 | 李振良（阿良），微信：k8init |
| 阿良教育官网 | [http://www.aliangedu.cn](http://www.aliangedu.cn/) |
| 说明 | 该文档有导航窗格，方便阅读，如果左侧没有显示，请检查word是否启用。  内部学员资料，请勿随意转载。 |



# 引言

## 1.1 项目背景

## 1.2 发布系统介绍

### 1.2.1 什么是发布系统

将应用自动上线。

### 1.2.2 应用自动发布的前提条件

标准化：线上服务器配置、路径、服务等统一标准

规范化：所有项目在整个发布流程都按统一的规范执行

### 1.2.3 发布环境

开发环境：开发人员自己测试的环境

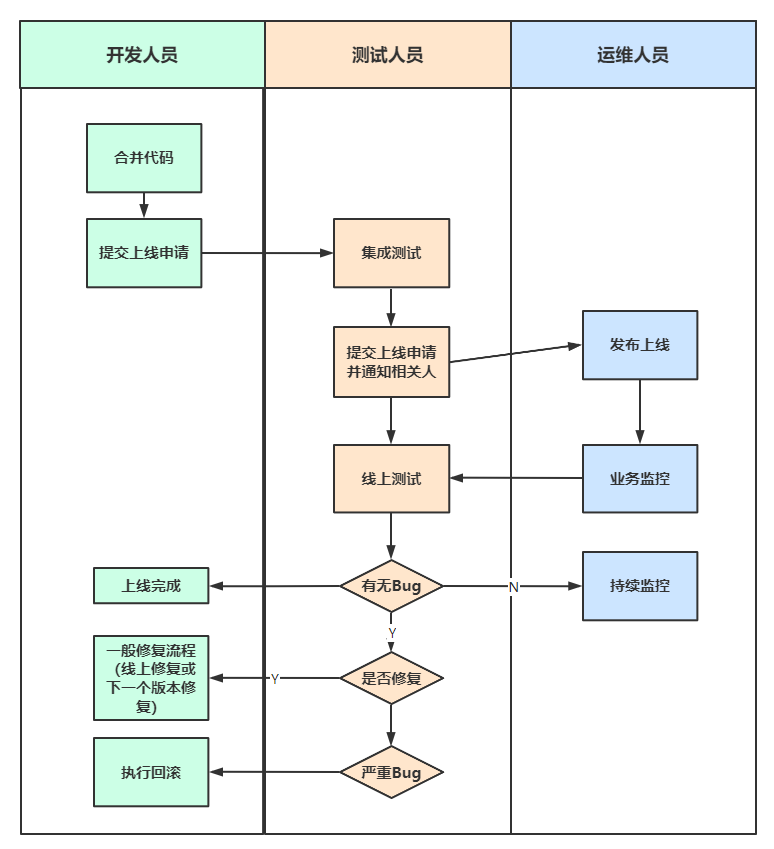
测试环境：测试人员测试的环境

生产环境：对互联网用户提供的环境

### 1.2.4 企业发布流程

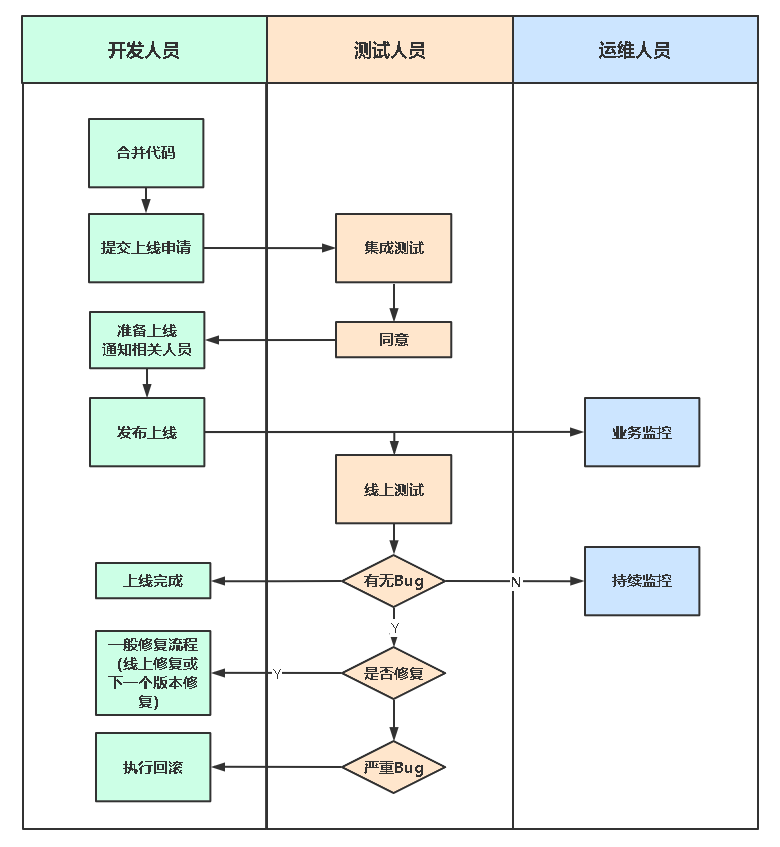
**小公司发布流程：**

* 开发人员提交代码，测试人员集成测试
* 测试人员测试通过，负责人合并代码到主分支
* 运维人员启动发布代码

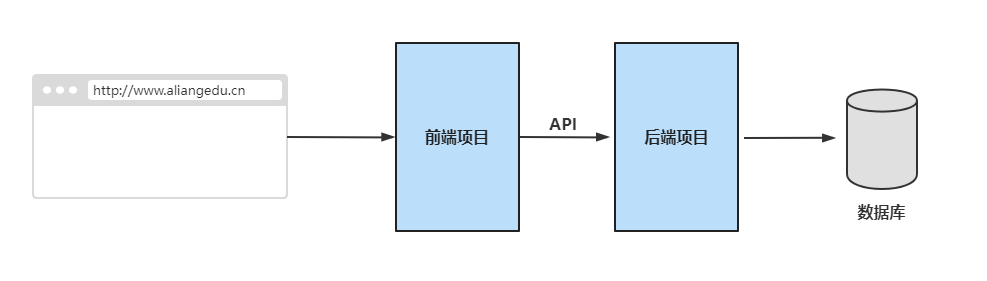


**大公司发布流程：**

* 开发人员提交代码，提交上线申请，测试人员集成测试
* 测试人员测试通过，负责人合并代码到主分支
* 开发人员启动发布代码



## 1.3 技术选型



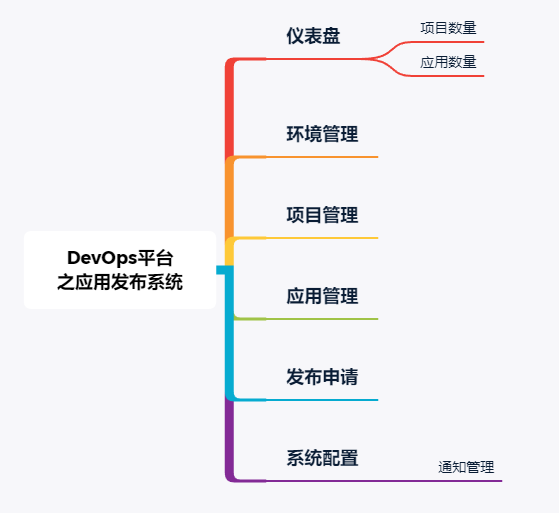
前端项目技术栈

* Vue
* Vue-router
* Element Plus
* Axios
* Echarts

后端项目技术栈

* Python
* Django DRF
* Mysql

## 1.4 整体设计



# 2. 数据库设计



## 2.1 环境管理

表名：app\_release\_env

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 空 | 名称 |
| id | INTEGER | 否 | 自增长ID |
| name | VARCHAR(30) | 否 | 环境名称 |
| english\_name | VARCHAR(30)（unique） | 否 | 英文名称 |
| note | TEXT | 是 | 备注 |
| create\_time | DATETIME | 否 | 创建时间 |

## 2.2 项目管理

表名：app\_release\_project

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 空 | 名称 |
| id | INTEGER | 否 | 自增长ID |
| name | VARCHAR(30) | 否 | 项目名称 |
| english\_name | VARCHAR(30)（unique） | 否 | 英文名称 |
| note | TEXT | 是 | 备注 |
| create\_time | DATETIME | 否 | 创建时间 |

## 2.3 应用管理

表名：app\_release\_app

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 空 | 名称 |
| id | INTEGER | 否 | 自增长ID |
| name | VARCHAR(30) | 否 | 应用名称 |
| english\_name | VARCHAR(30)（unique） | 否 | 英文名称 |
| project | 项目表一对多关系 | 否 | 项目名称 |
| create\_time | DATETIME | 否 | 创建时间 |

### 2.3.1 发布配置（新建与查看）

表名：app\_release\_config

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 空 | 名称 |
| id | INTEGER | 否 | 自增长ID |
| app | 应用表一对多关系 | 否 | 应用 |
| env | 环境表一对多关系 | 否 | 发布环境 |
| server\_ids | JSON | 否 | 目标主机（列表），调用/cmdb/server接口 |
| git\_repo | VARCHAR(200) | 否 | Git仓库 |
| git\_credential\_id | INTEGER | 是 | Git凭据，调用/config/credential接口 |
| notify\_id | INTEGER | 是 | 消息通知，调用/config/[notification](javascript:;)接口 |
|  |  |  |  |
| exclude\_files | TEXT | 是 | 部署排除文件 |
| global\_variables | TEXT | 是 | 自定义全局变量 |
| pre\_checkout\_script | TEXT | 是 | 代码检出前执行脚本 |
| post\_checkout\_script | TEXT | 是 | 代码检出后执行脚本 |
| dst\_dir | VARCHAR(50) | 否 | 部署路径 |
| history\_version\_dir | VARCHAR(50) | 否 | 历史版本路径 |
| history\_version\_number | INTEGER | 是 | 历史版本保留数 |
| pre\_deploy\_script | TEXT | 是 | 应用发布前执行脚本 |
| post\_deploy\_script | TEXT | 是 | 应用发布后执行脚本 |
| create\_time | DATETIME | 否 | 创建时间 |

## 2.4 发布申请

表名：app\_release\_apply

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 空 | 名称 |
| release\_config | 发布配置表一对多关系 |  | 发布配置 |
| title | VARCHAR(30) | 否 | 申请标题 |
| branch | VARCHAR(100) | 是 | 代码分支 |
| job\_name | VARCHAR(30) | 否 | jenkins项目名，格式：环境-项目-应用名称 |
| server\_list | JSON | 否 | 目标主机 |
| status | VARCHAR(30) | 否 | 发布状态（待发布、发布中、发布成功、发布异常） |
| note | VARCHAR(30) | 是 | 备注 |
| release\_time | DATETIME | 否 | 发布时间 |
| create\_time | DATETIME | 否 | 创建时间 |

注：前端所需环境和应用从发布配置里获取

jenkins与目标主机提前做好ssh免交互。

## 2.5 系统配置：通知管理

表名：system\_config\_notify

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 类型 | 空 | 名称 |
| id | INTEGER | 否 | 自增长ID |
| notify\_mode | VARCHAR(20) | 否 | 通知方法（邮箱，钉钉，企业微信，手机号） |
| email\_recipient | VARCHAR(200) | 是 | 邮箱收件人列表 |
| dingding\_webhook | VARCHAR(100) | 是 | 钉钉Webhook |
| weixin\_webhook | VARCHAR(100) | 是 | 企业微信Webhook |
| note | TEXT | 是 | 备注 |
| create\_time | DATETIME | 否 | 创建时间 |

# 3. API平台开发

## 3.1 接口设计

|  |  |  |  |
| --- | --- | --- | --- |
| 请求路径 | http方法 | 功能 | 备注 |
| /api/app\_release/env | get,post,put,delete | 查看，创建，更新，删除 | 环境管理 |
| /api/app\_release/project | get,post,put,delete | 查看，创建，更新，删除 | 项目管理 |
| /api/app\_release/app | get,post,put,delete | 查看，创建，更新，删除 | 应用管理 |
| /api/app\_release/config | get,post,put,delete | 查看，创建，更新，删除 | 发布配置 |
| /api/app\_release/apply | get,post,put,delete | 查看，创建，更新，删除 | 发布申请 |
| /api/app\_release/deploy | post | 发布 | 调用ansible |
| /api/app\_release/git | get | 获取 | 获取分支 |
| /api/app\_release/rollback | get，post | 查看，回滚 |  |

## 3.2 API平台雏形

创建应用：

python manage.py startapp app\_release

添加APP：

INSTALLED\_APPS = [  
 ...  
 'rest\_framework',  
 'app\_release',  
]

### 1、数据库模型

from django.db import models  
  
class Env(models.Model):  
 name = models.CharField(max\_length=30, verbose\_name="环境名称")  
 english\_name = models.CharField(max\_length=30, unique=True, verbose\_name="英文名称")  
 note = models.TextField(null=True, blank=True, verbose\_name="备注")  
 create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="创建时间")  
  
 class Meta:  
 db\_table = 'app\_release\_env'  
 verbose\_name\_plural = '环境管理'  
 ordering = ('-id',)  
 def \_\_str\_\_(self):  
 return self.name  
  
class Project(models.Model):  
 name = models.CharField(max\_length=30, verbose\_name="项目名称")  
 english\_name = models.CharField(max\_length=30, unique=True, verbose\_name="英文名称")  
 note = models.TextField(null=True, blank=True, verbose\_name="备注")  
 create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="创建时间")  
  
 class Meta:  
 db\_table = 'app\_release\_project'  
 verbose\_name\_plural = '项目管理'  
 ordering = ('-id',)  
 def \_\_str\_\_(self):  
 return self.name  
  
class App(models.Model):  
 name = models.CharField(max\_length=30, verbose\_name="应用名称")  
 english\_name = models.CharField(max\_length=30, unique=True, verbose\_name="英文名称")  
 project = models.ForeignKey(Project, on\_delete=models.PROTECT, verbose\_name="所属项目")  
 create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="创建时间")  
  
 class Meta:  
 db\_table = 'app\_release\_app'  
 verbose\_name\_plural = '应用管理'  
 ordering = ('-id',)  
 def \_\_str\_\_(self):  
 return self.name  
  
class ReleaseConfig(models.Model):  
 app = models.ForeignKey(App, on\_delete=models.PROTECT, verbose\_name="应用名称")  
 env = models.ForeignKey(Env, on\_delete=models.PROTECT, verbose\_name="发布环境")  
 server\_ids = models.JSONField(max\_length=100, verbose\_name="目标主机")  
 git\_repo = models.CharField(max\_length=100, verbose\_name="Git仓库")  
 git\_credential\_id = models.IntegerField(null=True, blank=True, verbose\_name="凭据ID")  
 notify\_id = models.IntegerField(null=True, blank=True,verbose\_name="消息通知ID")  
 note = models.TextField(null=True, blank=True, verbose\_name="备注")  
  
 exclude\_files = models.TextField(null=True, blank=True, verbose\_name="部署排除文件")  
 global\_variables = models.TextField(null=True, blank=True, verbose\_name="自定义全局变量")  
 pre\_checkout\_script = models.TextField(null=True, blank=True, verbose\_name="代码检出前执行脚本")  
 post\_checkout\_script = models.TextField(null=True, blank=True, verbose\_name="代码检出后执行脚本")  
  
 dst\_dir = models.CharField(max\_length=100, verbose\_name="服务器部署路径")  
 history\_version\_dir = models.CharField(max\_length=100, verbose\_name="历史版本路径")  
 history\_version\_number = models.IntegerField(default=7, verbose\_name="历史版本保留数")  
 pre\_deploy\_script = models.TextField(null=True, blank=True, verbose\_name="部署前执行脚本")  
 post\_deploy\_script = models.TextField(null=True, blank=True, verbose\_name="部署后执行脚本")  
  
 create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="创建时间")  
  
 class Meta:  
 db\_table = 'app\_release\_config'  
 verbose\_name\_plural = '发布配置'  
 ordering = ('-id',)  
 def \_\_str\_\_(self):  
 return self.app  
  
class ReleaseApply(models.Model):  
 release\_config = models.ForeignKey(ReleaseConfig, on\_delete=models.PROTECT, verbose\_name="发布配置")  
 title = models.CharField(max\_length=30, verbose\_name="发布标题")  
 branch = models.CharField(max\_length=100, verbose\_name="代码分支")  
 server\_ids = models.JSONField(max\_length=100, verbose\_name="目标主机")  
 version\_id = models.CharField(max\_length=30, unique=True, verbose\_name="版本标识") # 自动生成，例如应用名称-分支-时间  
 status\_choice = (  
 (1, "待发布"),  
 (2, "发布中"),  
 (3, "发布成功"),  
 (4, "发布异常")  
 )  
 status = models.IntegerField(choices=status\_choice, default=1, verbose\_name="发布状态")  
 note = models.TextField(null=True, blank=True, verbose\_name="备注")  
 create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="创建时间")  
 release\_time = models.DateTimeField(auto\_now=True, verbose\_name="发布时间")  
  
 class Meta:  
 db\_table = 'app\_release\_apply'  
 verbose\_name\_plural = '发布申请'  
 ordering = ('-id',)  
 def \_\_str\_\_(self):  
 return self.title

消息通知：

class Notify(models.Model):  
 notify\_choice = (  
 (1, "邮件"),  
 (2, "钉钉"),  
 (3, "企业微信")  
 )  
 name = models.CharField(max\_length=50, verbose\_name="名称")  
 notify\_mode = models.IntegerField(choices=notify\_choice, default=1, verbose\_name="通知方式")  
 email\_recipient = models.CharField(max\_length=200, null=True, blank=True, verbose\_name="邮件收件人")  
 dingding\_webhook = models.CharField(max\_length=100, null=True, blank=True, verbose\_name="钉钉Webhook")  
 weixin\_webhook = models.CharField(max\_length=100, null=True, blank=True, verbose\_name="企业Webhook")  
 note = models.TextField(null=True, blank=True, verbose\_name="备注")  
 create\_time = models.DateTimeField(auto\_now\_add=True, verbose\_name="创建时间")  
  
 class Meta:  
 db\_table = "system\_config\_notify"  
 verbose\_name\_plural = "通知管理"  
 ordering = ('-id',)  
  
 def \_\_str\_\_(self):  
 return self.name

### 2、序列化

from app\_release.models import Env, Project, App, ReleaseConfig, ReleaseApply  
from rest\_framework import serializers  
  
class EnvSerializer(serializers.ModelSerializer):  
 """  
 发布环境序列化类  
 """  
 class Meta:  
 model = Env  
 fields = "\_\_all\_\_"  
 read\_only\_fields = ("id",) # 仅用于序列化（只读）字段，反序列化（更新）可不传  
  
class ProjectSerializer(serializers.ModelSerializer):  
 """  
 项目序列化类  
 """  
 class Meta:  
 model = Project  
 fields = "\_\_all\_\_"  
 read\_only\_fields = ("id",)  
  
class AppSerializer(serializers.ModelSerializer):  
 """  
 服务器序列化类  
 """  
 project = ProjectSerializer()  
  
 class Meta:  
 model = App  
 fields = "\_\_all\_\_"  
 read\_only\_fields = ("id", )  
  
class ReleaseConfigSerializer(serializers.ModelSerializer):  
 """  
 发布配置序列化类  
 """  
 app = AppSerializer()  
 env = EnvSerializer()  
  
 class Meta:  
 model = ReleaseConfig  
 fields = "\_\_all\_\_"  
 read\_only\_fields = ("id","create\_time")  
  
class ReleaseApplySerializer(serializers.ModelSerializer):  
 """  
 发布申请序列化类  
 """  
 release\_config = ReleaseConfigSerializer()  
  
 class Meta:  
 model = ReleaseApply  
 fields = "\_\_all\_\_"  
 read\_only\_fields = ("id",)

### 3、视图

from rest\_framework.viewsets import ModelViewSet  
from rest\_framework.views import APIView  
from rest\_framework.response import Response  
from app\_release.models import Env, Project, App, ReleaseConfig, ReleaseApply  
from app\_release.serializers import EnvSerializer, ProjectSerializer, AppSerializer, ReleaseConfigSerializer, ReleaseApplySerializer  
from rest\_framework import filters  
from django\_filters.rest\_framework import DjangoFilterBackend  
  
class EnvViewSet(ModelViewSet):  
 queryset = Env.objects.all()  
 serializer\_class = EnvSerializer  
 filter\_backends = [filters.SearchFilter, filters.OrderingFilter, DjangoFilterBackend] # 指定过滤器  
 search\_fields = ('name',) # 指定可搜索字段  
 filter\_fields = ('name',) # 指定过滤字段  
  
class ProjectViewSet(ModelViewSet):  
 queryset = Project.objects.all()  
 serializer\_class = ProjectSerializer  
 filter\_backends = [filters.SearchFilter, filters.OrderingFilter, DjangoFilterBackend]   
 search\_fields = ('name',)  
 filter\_fields = ('name',)  
  
class AppViewSet(ModelViewSet):  
 queryset = App.objects.all()  
 serializer\_class = AppSerializer  
 filter\_backends = [filters.SearchFilter, filters.OrderingFilter, DjangoFilterBackend]   
 search\_fields = ('name',)  
 filter\_fields = ('project',)  
  
class ReleaseConfigViewSet(ModelViewSet):  
 queryset = ReleaseConfig.objects.all()  
 serializer\_class = ReleaseConfigSerializer  
  
class ReleaseApplyViewSet(ModelViewSet):  
 queryset = ReleaseApply.objects.all()  
 serializer\_class = ReleaseApplySerializer  
 filter\_backends = [filters.SearchFilter, filters.OrderingFilter, DjangoFilterBackend]   
 search\_fields = ('title',)  
 filter\_fields = ('status',)

### 4、路由

from django.contrib import admin  
from django.urls import path, include, re\_path  
from libs import token\_auth  
from rest\_framework\_swagger.views import get\_swagger\_view  
schema\_view = get\_swagger\_view(title='接口文档')  
  
from cmdb.views import ExcelCreateHostView, CreateHostView, HostCollectView, TencentCloudView, AliyunCloudView  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 re\_path('^api/login/$', token\_auth.CustomAuthToken.as\_view()),  
 re\_path('^api/change\_password/$', token\_auth.ChangeUserPasswordView.as\_view()),  
 re\_path('^api/docs/$', schema\_view),  
 re\_path('^api/cmdb/create\_host/$', CreateHostView.as\_view()),  
 re\_path('^api/cmdb/excel\_create\_host/$', ExcelCreateHostView.as\_view()),  
 re\_path('^api/cmdb/host\_collect/$', HostCollectView.as\_view()),  
 re\_path('^api/cmdb/tencent\_cloud/$', TencentCloudView.as\_view()),  
 re\_path('^api/cmdb/aliyun\_cloud/$', AliyunCloudView.as\_view()),  
]  
  
# CMDB系统  
from cmdb.views import IdcViewSet, ServerGroupViewSet, ServerViewSet, ExcelCreateHostView  
from rest\_framework import routers  
router = routers.DefaultRouter()  
router.register(r'cmdb/idc', IdcViewSet, basename="idc")  
router.register(r'cmdb/server\_group', ServerGroupViewSet, basename="server\_group")  
router.register(r'cmdb/server', ServerViewSet, basename="server")  
  
# 发布系统  
from app\_release.views import EnvViewSet, ProjectViewSet, AppViewSet, ReleaseConfigViewSet, ReleaseApplyViewSet  
router.register(r'app\_release/env', EnvViewSet, basename="env")  
router.register(r'app\_release/project', ProjectViewSet, basename="project")  
router.register(r'app\_release/app', AppViewSet, basename="app")  
router.register(r'app\_release/config', ReleaseConfigViewSet, basename="config")  
router.register(r'app\_release/apply', ReleaseApplyViewSet, basename="apply")  
  
# 系统配置  
from system\_config.views import CredentialViewSet  
router.register(r'config/credential', CredentialViewSet, basename="credential")  
  
urlpatterns += [  
 path('api/', include(router.urls))  
]

## 3.3 发布申请

### 获取Git分支

class GitView(APIView):  
 def get(self, request):  
 git\_repo = request.query\_params.get('git\_repo')  
  
 from git.repo import Repo  
 import os, shutil  
  
 clone\_dir = os.path.join(settings.BASE\_DIR, 'repo')  
  
 if os.path.exists(clone\_dir):  
 shutil.rmtree(clone\_dir)  
  
 Repo.clone\_from(git\_repo, to\_path=clone\_dir)  
 # 获取分支  
 repo = Repo(clone\_dir)  
 branch = []  
 for ref in repo.remote().refs:  
 if ref.remote\_head != 'HEAD':  
 branch.append(ref.remote\_head)  
  
 res = {'code': 200, 'msg': '获取成功', 'data': branch}  
 return Response(res)

## 3.4 发布

## 3.5 回滚

# 2. 前端开发

## 2.1 环境管理、项目管理

与机房管理类似，需修改：

* 修改路由
* 修改接口地址
* 修改字段
* 修改展示列
* 修改对话框标题
* 修改子组件名称

## 2.2 应用管理

### 新建发布（子组件）

### 发布配置（子组件）

## 2.3 发布申请

### 创建表单



* 发布环境与发布应用联动，必须先选择发布环境。

// 获取发布环境  
getEnv() {  
 this.$http.get('/app\_release/env/')  
 .then(res => {  
 this.env = res.data.data;  
 });  
 this.createForm.app = ''; // 再次选择发布环境将发布应用表单清空  
},

获取发布应用从关联的发布配置中获取：

// 获取发布配置  
getReleaseConfig() {  
 if(this.form.env) {  
 // 获取所有发布配置  
 this.$http.get('/app\_release/config/', {params: {'env': this.form.env}}) // this.createForm.env 环境ID  
 .then(res => {  
 this.releaseConfig = res.data.data; // 临时存储所有应用发布配置  
 });  
 } else {  
 this.$message.warning('请先选择发布环境！')  
 }  
},

请求发布配置接口过滤指定环境。

例如 http://127.0.0.1:8001/api/app\_release/config/?env=2

但env字段是个关系表，不能直接配置过滤，需要自己写过滤规则。



serializers.py

from django\_filters import rest\_framework as filters  
class ReleaseConfigFilter(filters.FilterSet):  
 env = filters.CharFilter(field\_name="env\_\_id") # 反向查询  
 class Meta:  
 model = ReleaseConfig  
 fields = ['env',]

views.py

class ReleaseConfigViewSet(ModelViewSet):  
 queryset = ReleaseConfig.objects.all()  
 serializer\_class = ReleaseConfigSerializer  
   
 # 指定自定义过滤器  
 from .serializers import ReleaseConfigFilter  
 filterset\_class = ReleaseConfigFilter

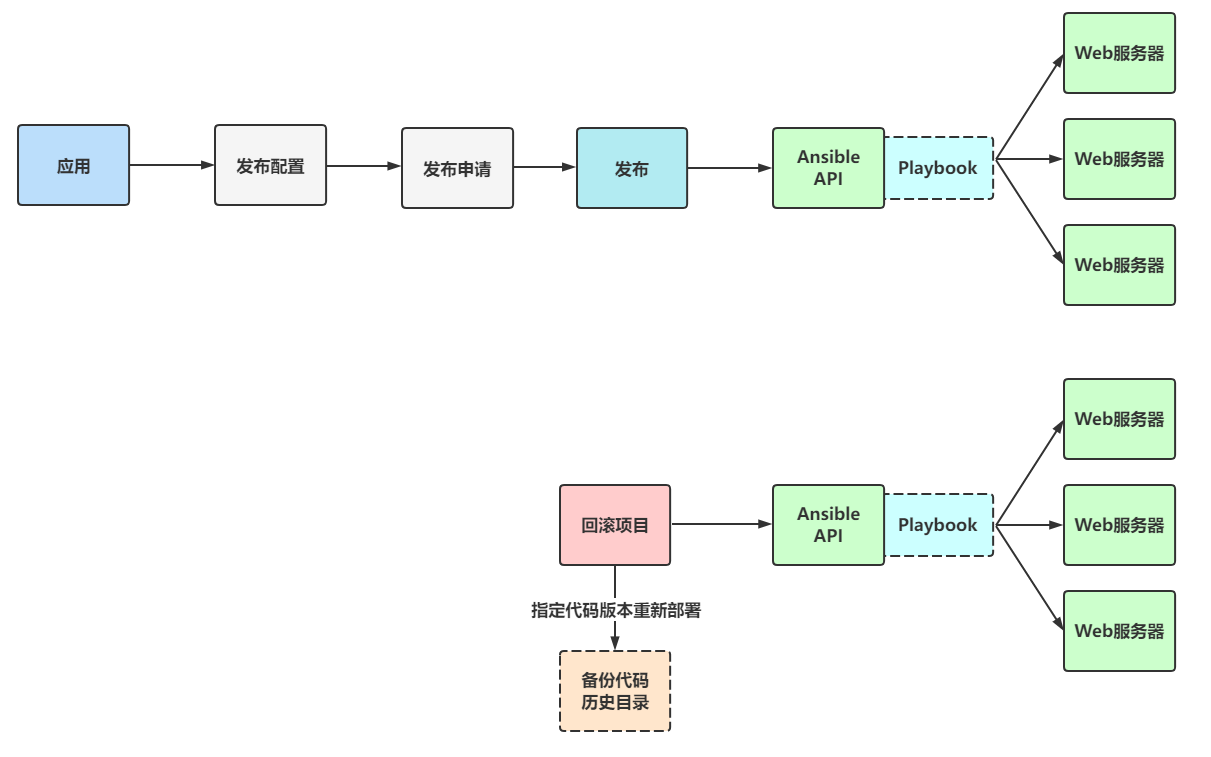
// 获取发布配置  
getReleaseConfig() {  
 if(this.form.env) {  
 // 获取所有发布配置  
 this.$http.get('/app\_release/config/', {params: {'env': this.form.env}}) // this.createForm.env 环境ID  
 .then(res => {  
 this.releaseConfig = res.data.data; // 临时存储所有应用发布配置  
 });  
 } else {  
 this.$message.warning('请先选择发布环境！')  
 }  
},

watch: {  
 // 当选择发布应用后，自动渲染目标主机  
 'form.app': {  
 handler: function () {  
 this.getServer(); // 执行获取服务器  
 this.releaseConfig.forEach(row => {  
 if(row.id == this.form.app) { // app是ID，获取选择发布应用对应发布配置里目标主机ID  
 this.form.server = row.server\_ids;  
 // 把发布配置id保存到里面，用于提交到后端  
 this.form.release\_config = row.id;  
 }  
 });  
 }  
 },  
}

提交：

def create(self, request, \*args, \*\*kwargs):  
 print(request.data)  
 release\_config\_id = request.data.get('release\_config')  
 title = request.data.get('title')  
 branch = request.data.get('branch')  
  
 # 版本唯一版本号  
 release\_config\_obj = ReleaseConfig.objects.get(id=release\_config\_id)  
 app\_name = release\_config\_obj.app.english\_name  
 from datetime import datetime  
 timestamp = datetime.now().strftime('%Y%m%d%H%M%S')  
 version\_id = "%s-%s-%s" %(app\_name, branch, timestamp) # 自动生成，例如应用名称-分支-时间  
  
 server\_ids = request.data.get('server')  
 status = 1 # 发布状态  
 note = request.data.get('note')  
  
 apply = ReleaseApply.objects.filter(version\_id=version\_id)  
 if not apply:  
 ReleaseApply.objects.create(  
 release\_config = release\_config\_obj,  
 title = title,  
 branch = branch,  
 version\_id = version\_id,  
 server\_ids = server\_ids,  
 status = status,  
 note = note  
 )  
 res = {'code': 200, 'msg': '创建成功'}  
 else:  
 res = {'code': 500, 'msg': '发布申请已存在！'}  
 return Response(res)

### 发布和回滚



### Ansible API

<https://docs.ansible.com/ansible/latest/dev_guide/developing_api.html>

windows支持ansible有问题，最好linux上测试：

yum install epel-release -y

yum install python3 git -y

pip3 install -U pip setuptools

pip3 install ansible -i <https://mirrors.aliyun.com/pypi/simple>

pip3 install -r requirements.txt -i <https://mirrors.aliyun.com/pypi/simple>

requirements.txt

aliyun-python-sdk-ecs==4.10.0

ansible

channels==2.4.0

Django==3.2

django-cors-headers==3.10.1

django-filter==2.4.0

django-rest-swagger==2.2.0

djangorestframework==3.13.0

GitPython==3.1.20

paramiko==2.9.2

PyMySQL==1.0.2

PyYAML==5.4.1

requests==2.27.1

tencentcloud-sdk-python==3.0.575

xlrd==1.2.0

playbook示例：

- name: 执行shell命令

hosts: webservers

gather\_facts: no

tasks:

- name: "查看ip"

shell: ip addr

## 2.5 消息通知

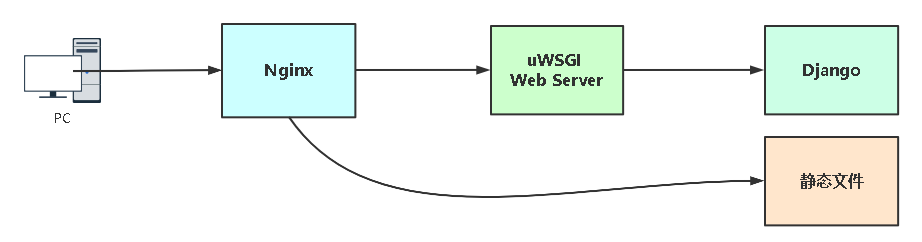
以钉钉为例：

import requests  
import json  
  
def dingtalk\_msg(content):  
 url = 'https://oapi.dingtalk.com/robot/send?access\_token=f41613ad17c00b7a43103391f7648f2b0409702cf26922669151467dc067c3c2'  
 headers = {'Content-Type': 'application/json;charset=utf-8'}  
 data = {  
 "msgtype": "text",  
 "text": {  
 "content": content  
 }  
 }  
 res = requests.post(url=url, headers=headers, data=json.dumps(data))  
 return res.text  
if \_\_name\_\_ == "\_\_main\_\_":  
 content = "发布成功"  
 print(dingtalk\_msg(content))

## 2.6 线上部署

### 后端

#### 1、部署架构



Nginx 前端Web服务，接收到动态请求通过uwsgi模块将请求转发给uwsgi服务器，uwsgi服务器通过django处理完后返回给Nginx，Nginx返回用户浏览器展示。

**既然uwsgi是一个可以独立部署的服务器，为什么还用Nginx代理？**

* Nginx作为入口可配置安全策略，并且可以为uwsgi提供负载均衡。
* Nginx处理静态资源能力强

#### 2、将本地开发的项目打包

**2.1 导出依赖模块列表**

pip3 freeze > requirements.txt

主要有以下模块：

Django==3.2

django-cors-headers==3.11.0

django-filter==21.1

django-rest-swagger==2.2.0

djangorestframework==3.12.4

GitPython==3.1.27

pandas==1.4.2

paramiko==2.9.2

PyMySQL==1.0.2

requests==2.27.1

xlrd==1.2.0

ansible==6.1.0

tencentcloud-sdk-python==3.0.575

aliyun-python-sdk-ecs==4.24.13

**2.2 修改数据库为Mysql**

指定数据库驱动：

# devops/\_\_init\_\_.py

import pymysql

pymysql.install\_as\_MySQLdb()

# vi devops/settings.py

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'k8s',

'USER': 'root',

'PASSWORD': '123456',

'HOST': '192.168.31.62',

'PORT': '3306',

}

}

关闭debug模式和白名单：

DEBUG = False # 调试模式

ALLOWED\_HOSTS = ['\*' ] # 白名单，只允许列表中的ip访问，\*代表所有

#### 3、服务器环境准备

##### 3.1 安装Python3

yum install zlib-devel libffi-devel mysql-devel bzip2-devel git -y

tar zxvf Python-3.8.6.tgz

cd Python-3.8.6

./configure

make && make install

##### 3.2 安装依赖模块列表

pip3 install -r requirements.txt -i https://mirrors.aliyun.com/pypi/simple

##### 3.3 安装数据库

MySQL:

docker run -d --name db -p 3306:3306 -v mysqldata:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD=123456 mysql:5.7 --character-set-server=utf8

docker exec -it db bash

root@e2eff2d75dd2:/# mysql -uroot -p$MYSQL\_ROOT\_PASSWORD -e "create database devops;"

启动开发环境，验证依赖模块：

python3 manage.py runserver 0.0.0.0:8080

测试问题，同步数据库：

python3 manage.py migrate

收集静态文件：

vi settings.py

STATIC\_URL = '/static/'

STATIC\_ROOT=os.path.join(BASE\_DIR,'static')

python3 manage.py collectstatic

创建管理员账号：

python3 manage.py createsuperuser

##### 3.4 安装与配置uwsgi

uWSGI是一个[Web服务器](https://baike.baidu.com/item/Web%E6%9C%8D%E5%8A%A1%E5%99%A8/8390210)，也是Python的一个模块，直接pip安装即可：

pip3 install uwsgi -i https://mirrors.aliyun.com/pypi/simple

创建uwsgi配置文件，路径任意：

mkdir /opt/devops\_api/uwsgi

# vi /opt/devops\_api/uwsgi/uwsgi.ini

[uwsgi]

# 项目目录

chdir = /opt/devops\_api

# 指定sock的文件路径

socket = /opt/devops\_api/uwsgi/uwsgi.sock

# 指定监听端口

# http = 0.0.0.0:8080

# 静态资源

static-map = /static=/opt/devops\_api/static

# wsgi文件（django入口）

wsgi-file=devops\_api/wsgi.py

# 进程个数

processes = 4

# 指定项目的应用

# module = devops\_api.wsgi

# 进程pid

pidfile = /opt/devops\_api/uwsgi/uwsgi.pid

# 日志路径

daemonize = /opt/devops\_api/uwsgi/uwsgi.log

启动：

vi /usr/lib/systemd/system/uwsgi.service

[Unit]

Description=HTTP Interface Server

[Service]

Type=forking

ExecStart=/usr/local/bin/uwsgi --ini /opt/devops\_api/uwsgi/uwsgi.ini

ExecReload=/bin/kill -s HUP $MAINPID

Restart=always

[Install]

WantedBy=multi-user.target

systemctl daemon-reload

systemctl start uwsgi

systemctl enable uwsgi

##### 3.5 配置Nginx

yum install epel-release –y

yum install nginx –y

vi /etc/nginx/nginx.conf

…

server {

listen 8080 default\_server;

server\_name \_;

location / {

include uwsgi\_params; # 导入模块用于与uwsgi通信

uwsgi\_pass unix:/opt/devops\_api/uwsgi/uwsgi.sock;

}

# 静态文件目录

location /static {

alias /opt/devops\_api/static;

}

}

systemctl start nginx

systemctl enable nginx

启动nginx，访问测试。

http://192.168.1.72:8080/admin

http://192.168.1.72:8080/api/app\_release/env/

### 前端

在项目目录下执行：

npm run build

会自动生成dist目录，放到/opt/devops\_web目录即可。

在nginx配置server：

server {

listen 80;

server\_name \_;

root /opt/devops\_web;

# 需要指向下面的@router否则会出现vue的路由在nginx中刷新出现404

location / {

try\_files $uri $uri /index.html;

}

}