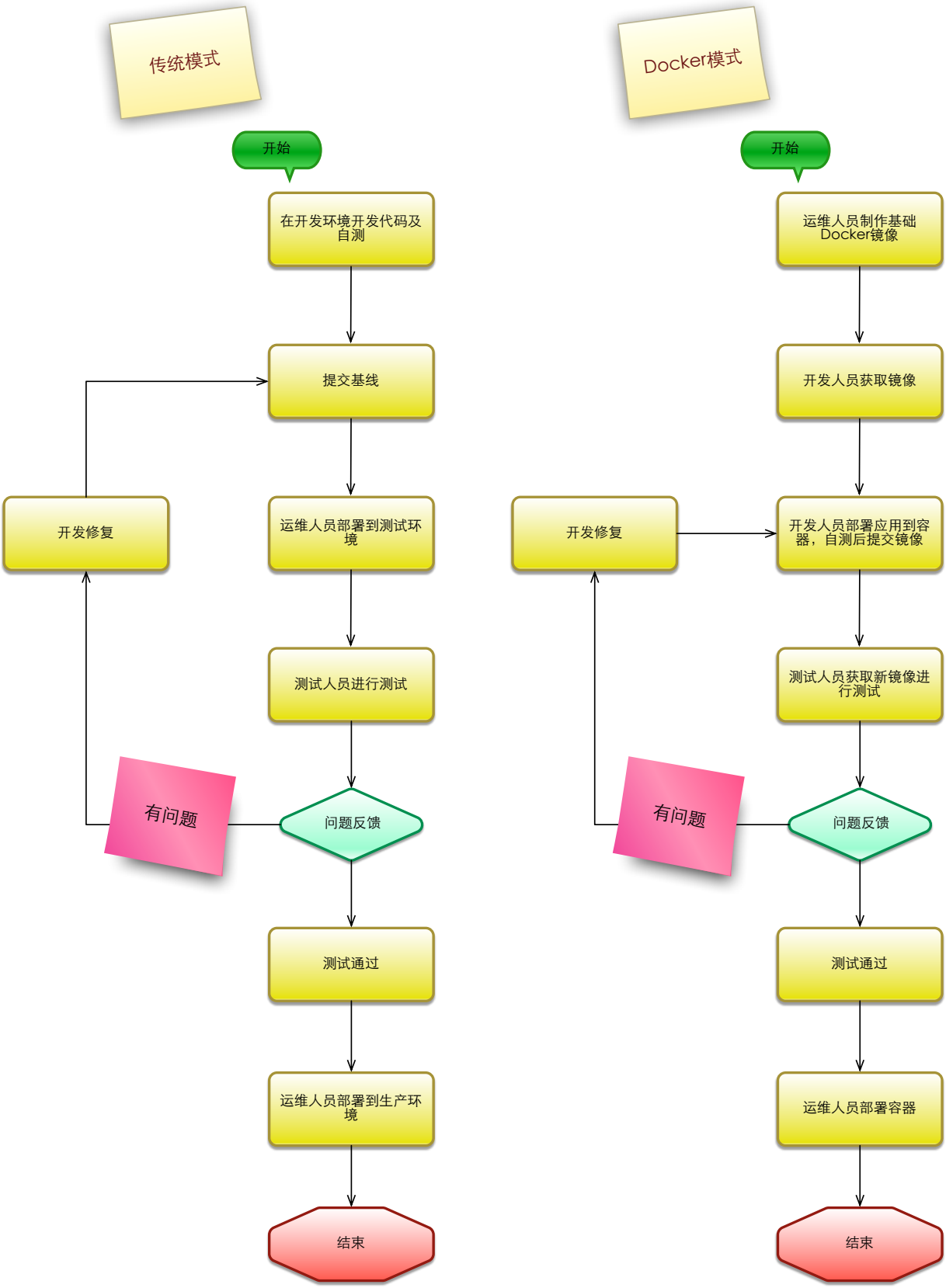
传统和Docker模式在软件部署方式上的比较

在传统模式中，开发团队在开发环境中完成软件开发，自己做了一遍单元测试，测试通过，提交到代码版本管理库，打包给 QA 进行测试。运维把应用部署到测试环境，QA 进行测试，没问题后通知部署人员发布到生产环境。

在上述过程中涉及到至少三个环境：开发、测试和生产。现实情况是，开发自测没问题，但到了测试或者生产环境程序无法运行，让开发团队排查，经过长时间排查最后发现是测试环境的一个第三方库过时了。这样的现象在软件开发中很普遍，已经不适用如今的快速开发和部署。

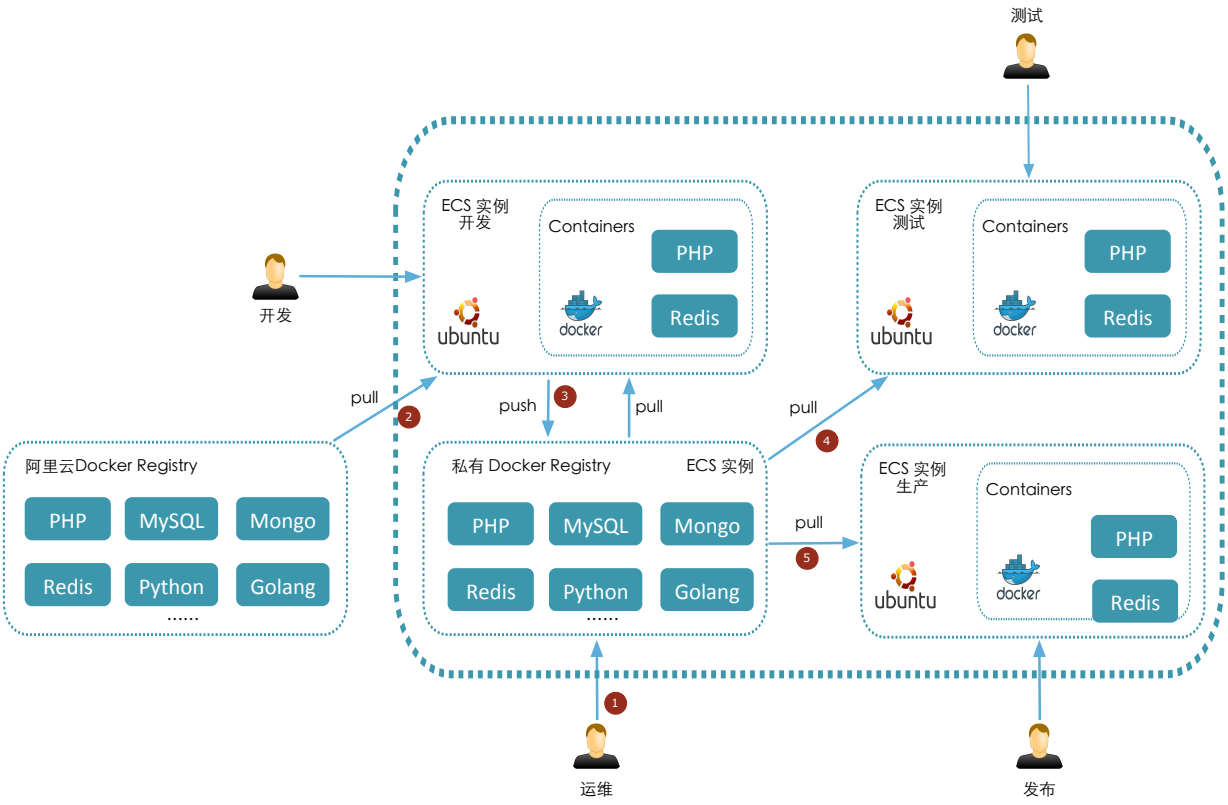
在 Docker 模式中，应用是以容器的形式存在，所有和该应用相关的依赖都会在容器中，因此移植非常方便，不会存在像传统模式那样的环境不一致。

下图描述了传统模式和 Docker 模式在软件部署方式上的区别：


传统模式
开始
在开发环境开发代码及自测
提交基线
运维人员部署到测试环境
测试人员进行测试
问题反馈
有问题
开发修复
测试通过
运维人员部署到生产环境
结束
Docker模式
开始
运维人员制作基础Docker镜像
开发人员获取镜像
开发人员部署应用到容器，自测后提交镜像
测试人员获取新镜像进行测试
问题反馈
有问题
开发修复
测试通过
运维人员部署容器
结束

ECS Docker 实践

下图描述了 ECS Docker 如何使开发、测试和运维之间合作更紧密，各司其职，效率更高。



以一个简单的应用开发、测试和发布来说明 Docker 在阿里云 ECS 上的运用：

1) 运维人员在 ECS 上搭建私有 Docker Registry。
2) 开发人员在开发 ECS 上从阿里云或私有 Docker Registry 获取应用需要的基础镜像。
3) 开发人员开发 ECS 上构造应用容器，自测后提交容器为新的镜像并推送到私有 Docker Registry，通知 QA 测试。
4) QA 在自己的测试 ECS 上启动容器，测试后，有问题则 a)，没问题则 b)。
   a) 通知开发修复，回到步骤 3)。
   b) 提交到私有 Docker Registry，准备发布。
5) 发布人员下载最新版本镜像并在生产 ECS 上启动容器。

下面以一个简单的 Guestbook(PHP+Redis)应用为例详细说明各步骤。

1. 搭建私有 Docker Registry

私有 Docker Registry 搭建非常简单，官方推荐的方式是直接使用 registry 镜像：

```
docker run -p 5000:5000 registry
```

也可以通过-e 参数来传递信息，如：

```
docker run -e STORAGE_PATH=/opt/docker/registry -p 5000:5000 registry
```

也可以使用阿里云 OSS 做后端存储，如:

```
docker run -e OSS_BUCKET=<your_ali_oss_bucket> -e
STORAGE_PATH=/docker/ -e OSS_KEY=<your_ali_oss_key> -e
OSS_SECRET=<your_ali_oss_secret> -p 5000:5000 -d chrisjin/registry
```

详细配置请参考 https://github.com/docker/docker-registry，用户如需要实现一些认证，可以搭建 Nginx 来做基础认证。

2. 在开发 ECS 上部署应用容器

2.1 制作 Docker Redis

2.1.1 创建 Redis 目录

```
mkdir redis
cd redis
```

2.1.2 制作 Dockerfile

```
vi Dockerfile
```

```
FROM centos:centos6
MAINTAINER bryan dev@dev.com

# install redis
RUN yum install -y curl tar make gcc wget

RUN cd /usr/local/src && \
       wget  http://download.redis.io/redis-stable.tar.gz  && \
       tar xf redis-stable.tar.gz && \
       cd redis-stable && \
       make && \
       make install

EXPOSE 6379
```

```
CMD ["/usr/local/bin/redis-server"]
```

MAINTAINER 是 Dockerfile 维护者信息，用户可以自定义。

2.1.3 构造 docker image

```
docker build -t="zubryan/redis" .

docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
zubryan/redis	latest	1f6680b24682	2 minutes ago	388.5 MB
centos	centos6	68edf809afe7	8 days ago	212.7 MB

“zubryan/redis”是镜像库名，用户可以自己指定。

2.1.4 运行 Redis 容器

```
docker run --name redis -p 6379:6379 -d zubryan/redis
docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
9418c0118373	bryan/redis:latest	/usr/local/bin/redis	2 seconds ago	Up 1 seconds	0.0.0.0:6379->6379/tcp	redis

2.1.5 验证 Redis 容器运行

在实例安装 redis 客户端后，运行以下命令验证 Redis 运行。

```
redis-cli

127.0.0.1:6379> set mykey "Hello"

OK

127.0.0.1:6379> get mykey

"Hello"
```

2.2 制作 Docker PHP 并链接至 Redis

2.2.1 创建 php-redis 目录

```
mkdir php-redis
cd php-redis
```

2.2.2 Guestbook 应用相关代码

Guestbook应用相关代码：

1) vi index.php

```
<?

set_include_path('.:/usr/share/php:/usr/share/pear:/vendor/predis');

error_reporting(E_ALL);
ini_set('display_errors', 1);

require 'predis/autoload.php';

if (isset($_GET['cmd']) === true) {
  header('Content-Type: application/json');
  if ($_GET['cmd'] == 'set') {
    $client = new Predis\Client([
      'scheme' => 'tcp',
      'host'   => getenv('DB_PORT_6379_TCP_ADDR'),
      'port'   => getenv('DB_PORT_6379_TCP_PORT'),
    ]);
    $client->set($_GET['key'], $_GET['value']);
    print('{"message": "Updated"}');
  } else {
    $read_port = getenv('DB_PORT_6379_TCP_PORT');

    if (isset($_ENV['DB_PORT_6379_TCP_PORT'])) {
      $read_port = getenv('DB_PORT_6379_TCP_PORT');
    }
    $client = new Predis\Client([
      'scheme' => 'tcp',
      'host'   => getenv('DB_PORT_6379_TCP_ADDR'),
      'port'   => $read_port,
    ]);

    $value = $client->get($_GET['key']);
    print('{"data": "' . $value . '"}');
  }
} else {
  phpinfo();
} ?>
```

2) vi controllers.js

```
var redisApp = angular.module('redis', ['ui.bootstrap']);

/**
 * Constructor
 */
function RedisController() {}

RedisController.prototype.onRedis = function() {
    this.scope_.messages.push(this.scope_.msg);
    this.scope_.msg = "";
    var value = this.scope_.messages.join();
    this.http_.get("/index.php?cmd=set&key=messages&value=" + value)
            .success(angular.bind(this, function(data) {
                this.scope_.redisResponse = "Updated.";
            }));
};

redisApp.controller('RedisCtrl', function ($scope, $http, $location) {
        $scope.controller = new RedisController();
        $scope.controller.scope_ = $scope;
        $scope.controller.location_ = $location;
        $scope.controller.http_ = $http;

        $scope.controller.http_.get("/index.php?cmd=get&key=messages")
            .success(function(data) {
                console.log(data);
                $scope.messages = data.data.split(",");
            });
});
```

3) vi index.html

```
<html ng-app="redis">
  <head>
    <title>Guestbook</title>
    <link rel="stylesheet"
href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css">
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.12/angular.min.js"></script>
    <script src="/controllers.js"></script>
    <script src="ui-bootstrap-tpls-0.10.0.min.js"></script>
  </head>
  <body ng-controller="RedisCtrl">
    <div style="width: 50%; margin-left: 20px">
      <h2>Guestbook</h2>
    <form>
    <fieldset>
    <input ng-model="msg" placeholder="Messages" class="form-control" type="text"
name="input"><br>
    <button type="button" class="btn btn-primary"
ng-click="controller.onRedis()">Submit</button>
    </fieldset>
    </form>
    <div>
      <div ng-repeat="msg in messages">
        {{msg}}
      </div>
```

```
    </div>
    </div>
  </body>
</html>
```

2.2.3 制作 php-redis Dockerfile

```
vi Dockerfile
  FROM brendanburns/php

  ADD index.php /var/www/index.php
  ADD controllers.js /var/www/controllers.js
  ADD index.html /var/www/index.html

  CMD /run.sh
```

2.2.4 构造 docker image

```
docker build -t="zubryan/php-redis" .

docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
zubryan/redis	latest	1f6680b24682	2 minutes ago	388.5 MB
brendanburns/php	latest	cff979880024	7 months ago	382.8 MB
zubryan/phpredis	latest	be6f7a78df1e	About a minute ago	382.8 MB
centos	centos6	68edf809afe7	8 days ago	212.7 MB

“zubryan/php-redis”是镜像库名，用户可以自己指定。

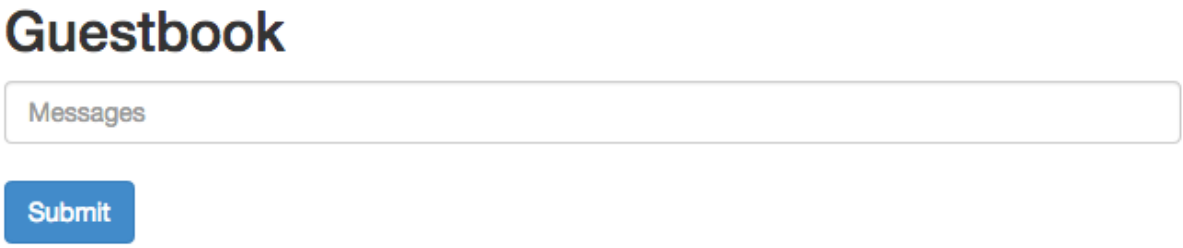
2.2.5 运行 php-redis 容器

运行php-redis容器，并链接之前运行的redis容器。

```
docker run --link redis:db -p 80:80 zubryan/php-redis
```

2.2.6 验证 php-redis 容器运行

打开浏览器，查看Guestbook应用。



3. 开发推送镜像到私有 Docker Registry

开发自测后，把 Redis 和 PHP image 推送到私有 Docker Registry，假定地址是 172.16.2.2。

```
docker tag zubryan/redis 172.16.2.2/zubryan/redis
docker push 172.16.2.2/zubryan/redis

docker tag zubryan/php-redis 172.16.2.2/zubryan/php-redis
docker push 172.16.2.2/zubryan/php-redis
```

4. 测试获取镜像在测试 ECS 上启动容器

```
docker run --name redis -p 6379:6379 -d 172.16.2.2/zubryan/redis
docker run --link redis:db -p 80:80 172.16.2.2/zubryan/php-redis
```

如果本地没有这两个镜像，会自动从私有 Docker Registry 拉过来。

5. 发布人员在生产 ECS 上启动容器

```
docker run --name redis -p 6379:6379 -d 172.16.2.2/zubryan/redis
docker run --link redis:db -p 80:80 172.16.2.2/zubryan/php-redis
```

如果本地没有这两个镜像，会自动从私有 Docker Registry 拉过来。