1年半運用してわかった！
BLUE PROTOCOLにおける
ビルドパイプラインのクラウド化の壁

株式会社バンダイナムコスタジオ

技術スタジオ 技術スタジオ付 エグゼクティブテクニカルディレクター
大井 隆義
総務ITサービス部
吉田 卓哉

BLUE PROTOCOL™

To save the world that is going to destroy, fight beyond the spacetime. Cooperate with friends, beat the mighty enemies, change the history. That is your mission. Now, let's run out! On a vast land, heading for a hopeful future!

blue-protocol.com

自己紹介

株式会社バンダイナムコスタジオ
技術スタジオ 技術スタジオ付
エグゼクティブテクニカルディレクター
大井 隆義

・2015年入社

BLUE PRTOCOLではエグゼクティブテクニカルディレクターとして技術全般をディレクション

実務としては主にグラフィック周りを担当。

株式会社バンダイナムコスタジオ
総務ITサービス部

吉田 卓哉

・2011年入社

・社内開発支援ツールの構築・管理・運用

・主に管理しているツール：
Perforce、Atlassian Confluence、JIRA、
Redmine、Incredibuild、Alienbrain…

・ここ数年でクラウド(主にAWS)を本格的に触り始める

本日お話する事(アジェンダ)

1. BLUE PROTOCOLとは？
2. ゲーム開発でのビルドパイプライン解説
3. BLUE PROTOCOLのビルドパイプラインクラウド化した話
4. クラウド化した結果や成果、わかった事
5. １年半運用した現状
6. 運用中に改善した事

アジェンダ

1. **BLUE PROTOCOLとは？**
2. ゲーム開発でのビルドパイプライン解説
3. BLUE PROTOCOLのビルドパイプラインクラウド化した話
4. クラウド化した結果や成果、わかった事
5. １年半運用した現状
6. 運用中に改善した事

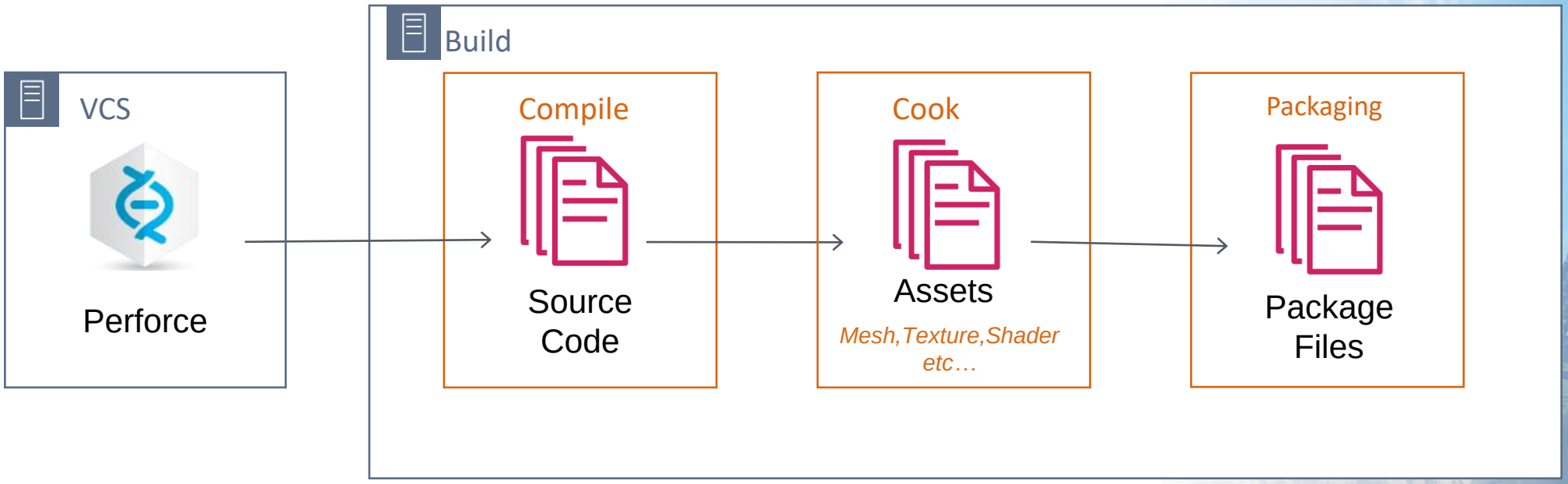
BLUE PROTOCOL(ブループロトコル)

- バンダイナムコオンライン、バンダイナムコスタジオで共同開発中のオンラインアクションRPG
- Unreal Engine 4で開発
- 2020年4月にCβT実施

アジェンダ

1. BLUE PROTOCOLとは？
2. **ゲーム開発でのビルドパイプライン解説**
3. BLUE PROTOCOLのビルドパイプラインクラウド化した話
4. クラウド化した結果や成果、わかった事
5. １年半運用した現状
6. 運用中に改善した事

ゲーム開発でのビルドパイプラインを解説



ゲーム開発でのビルドパイプラインを解説

ソースコードのコンパイル

- ソースコード（テキスト）から実行ファイル（exe）に
- 大量のソースコード
  - UnrealEngine4のコードだけでも10万ファイルを超える
  - ゲームのコードも数千ファイルはある
- IncrediBuildで分散してようやく

ゲーム開発でのビルドパイプラインを解説

アセットのクック

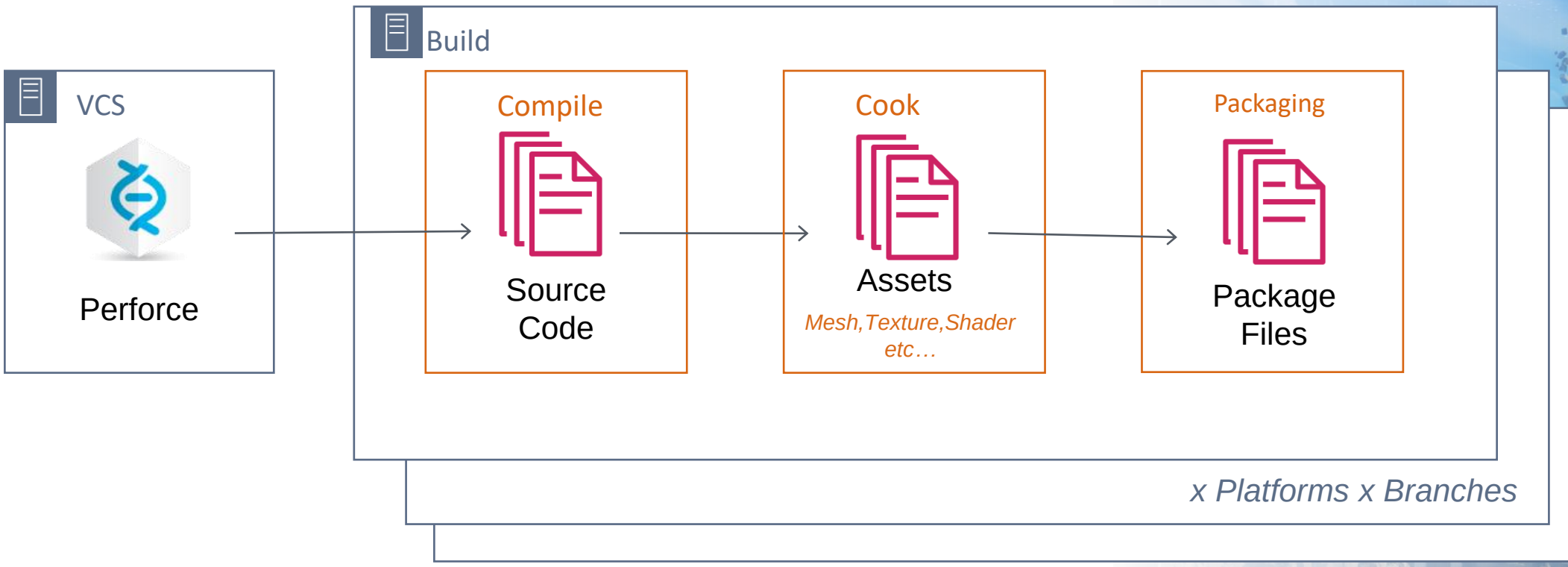
- 中間ファイルからそれぞれのプラットフォームに適したバイナリに
- 大量のアセット
  - モデル、テクスチャー、アニメーション、シェーダー etc..
  - 10万ファイルを越え、それぞれのサイズも大きい
- アセットの種類によっては分散できる
- キャッシュ利用である程度は軽減

ゲーム開発でのビルドパイプラインを解説

実機データをパッケージ

- クックされてできたバイナリを圧縮＆結合
- 最終的に結合されるデータをいくつかに分けることによって並列化

ゲーム開発でのビルドパイプラインを解説



ナイトリーで行っているパッケージング

プラットフォームやブランチなど
条件が増えれば増えるほどリソースが必要になる

アジェンダ

1. BLUE PROTOCOLとは？
2. ゲーム開発でのビルドパイプライン解説
3. **BLUE PROTOCOLのビルドパイプラインクラウド化した話**
4. クラウド化した結果や成果、わかった事
5. １年半運用した現状
6. 運用中に改善した事

ゲーム開発環境をクラウド化しようとすると・・・

ゲームサーバは既にクラウド運用しているので開発環境もクラウド化したい！

開発者

社外（パブリッククラウド）から社内ファイルサーバやインフラ、ツールへの通信は許可できません。

情報システム部

社内で使っているXXシステム、ライセンス契約でクラウドでの利用が禁止されています

システム管理者

ファーストパーティ製SDKをクラウドに置くのは先方の確認が取れないのでNG。

プロジェクトマネージャー

クラウド移行の際によく出る話題

■ **社内LAN前提で構築されたゲーム開発環境**
・ユーザー認証基盤, ファイルサーバ, VCS, インフラ,情報セキュリティ
,プロジェクトごとの開発プロセスごとのローカルルール

■ **ネットワークスピード・トラフィック・レイテンシの比較**
・大量・大容量アセットデータ → 1つのpsdファイルが1GB越え
・分散ビルド環境 → いつでも150コア並列分散ビルド
・描画性能(GPU)や操作レスポンス → 4k 60fps

■ **契約関連(NDA,EULA)**
・ゲームエンジン、ミドルウェア等の開発ツール
・1stパーティ製SDK
・キャラクターの知的財産(IP)

クラウド移行の際によく出る話題

- **社内LAN前提で構築されたゲーム開発環境**
  - ユーザー認証基盤、ファイルサーバ、VCS、インフラ、情報セキュリティ
  - プロジェクトごとの開発プロセスごとのローカルルール
- **ネットワークスピード・トラフィック・レイテンシ**
  - 大量・大容量アセットデータ → 1つのpsdファイルが1GB越え
  - 分散ビルド環境 → いつでも150コア並列分散ビルド
  - 描画性能(GPU)や操作レスポンス → 4k 60fps
- **契約関連(NDA,EULA)**
  - ゲームエンジン、ミドルウェア等の開発ツール
  - 1stパーティ製SDK

開発プロセスすべてをクラウド化するのは難易度・コスト共に高い

↓

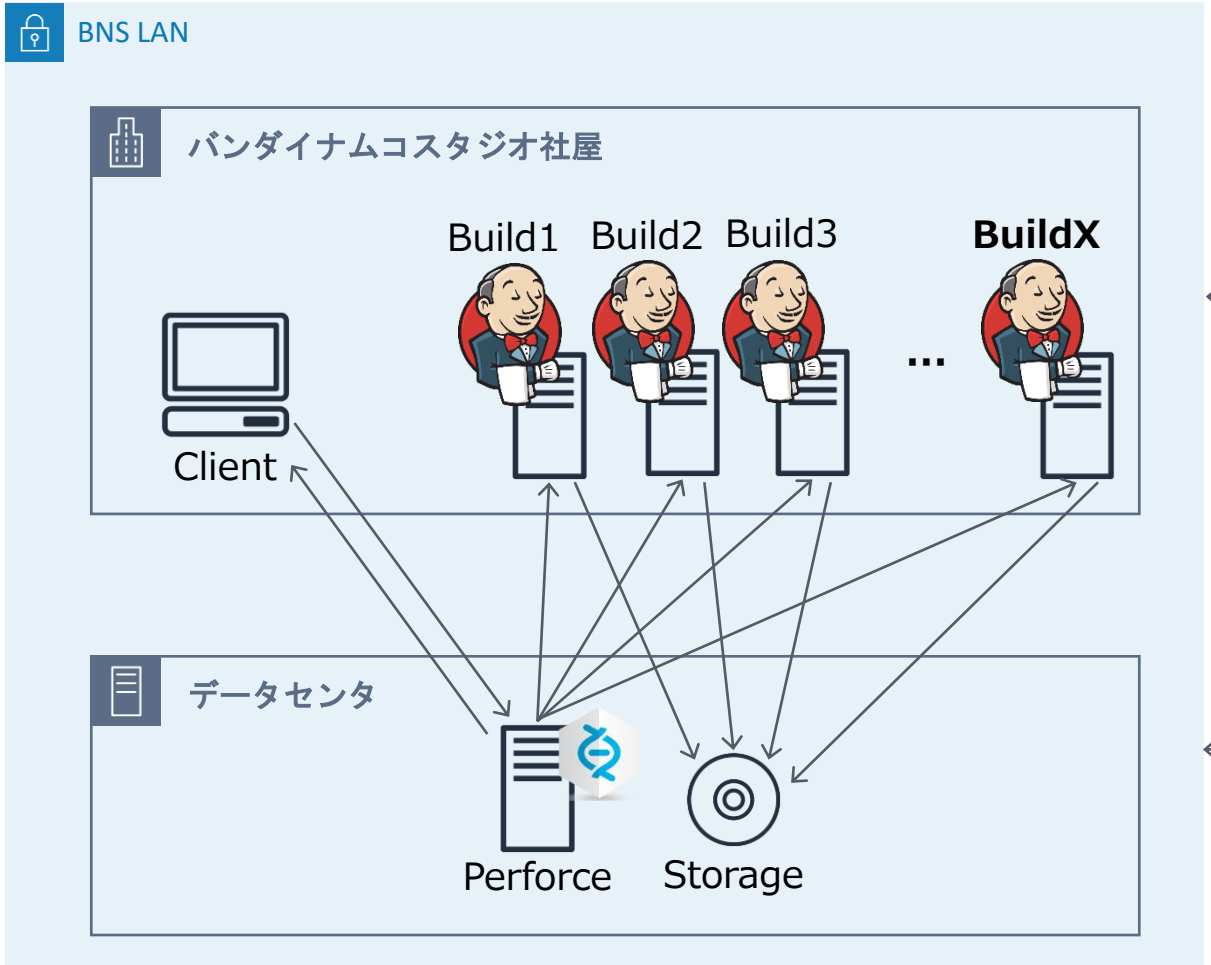
クラウド化しやすい部分からやってみる

↓

BLUE PROTOCOLのビルドパイプラインをクラウド化してみる

ビルドパイプライン：Before

従来のビルドパイプライン構成



アクセスNG

Firewall
(BNS)

アクセスNG

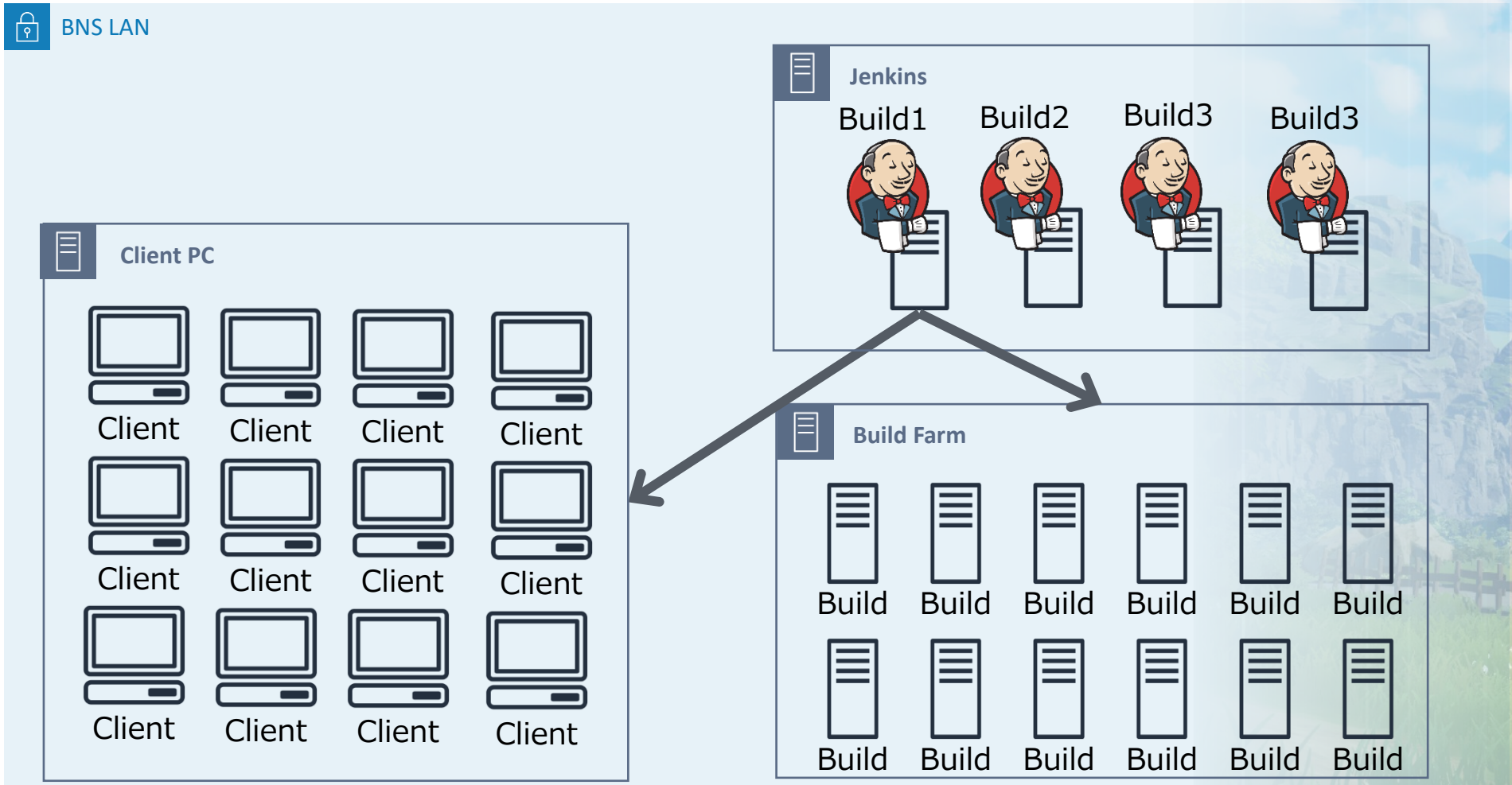
社外開発協力会社1
Client
Client

社外開発協力会社2
Client
Client

ビルドパイプライン：Before

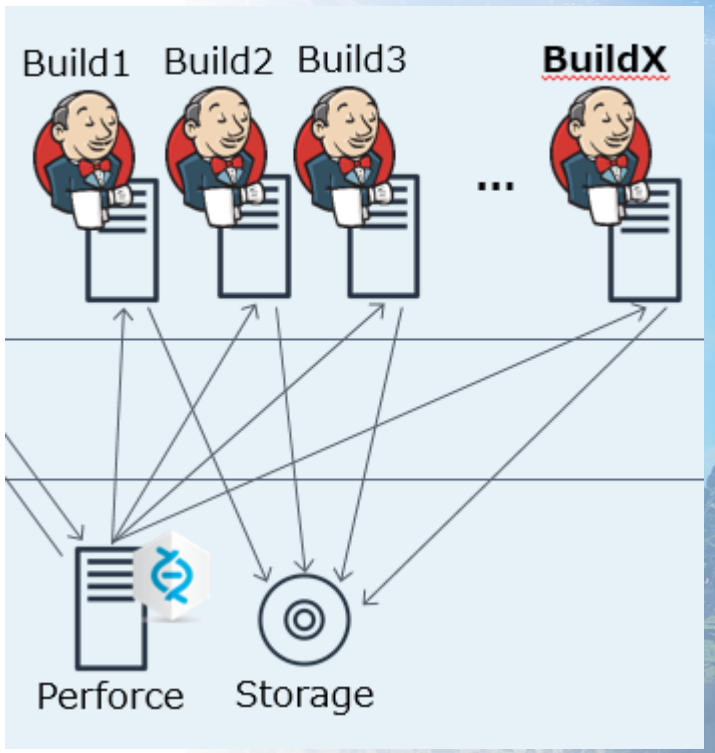
150コア使った並列分散ビルド

Incredibuildを使うことで開発者PC、ビルドファームの余剰CPUを使って分散ビルド



ビルドパイプライン：Before

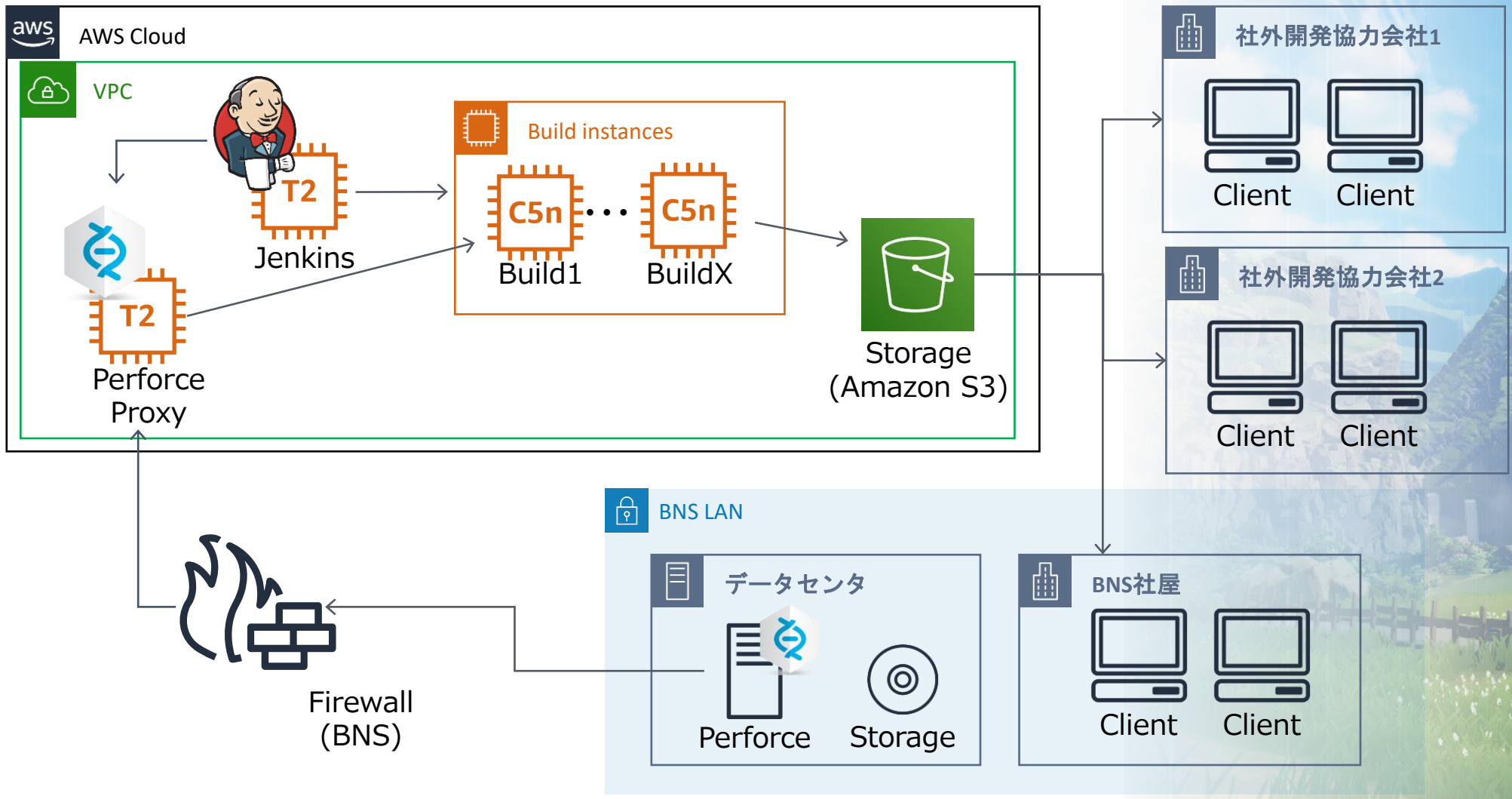
従来のビルドパイプラインの悩み



- ビルドマシンの調達に時間がかかる
  - ・2週間程度
- ビルドマシンのセットアップが大変
  - ・各種ツールの初期インストール
  - ・Jenkinsのジョブ設定
- ビルドマシンの運用が大変
  - ・設置スペース・停電対応・故障対応
- 増えるJenkins
  - ・運用期間が長くなるほど故障時のトラブル対応のリスクが大きくなる→忙しい時に限って故障する

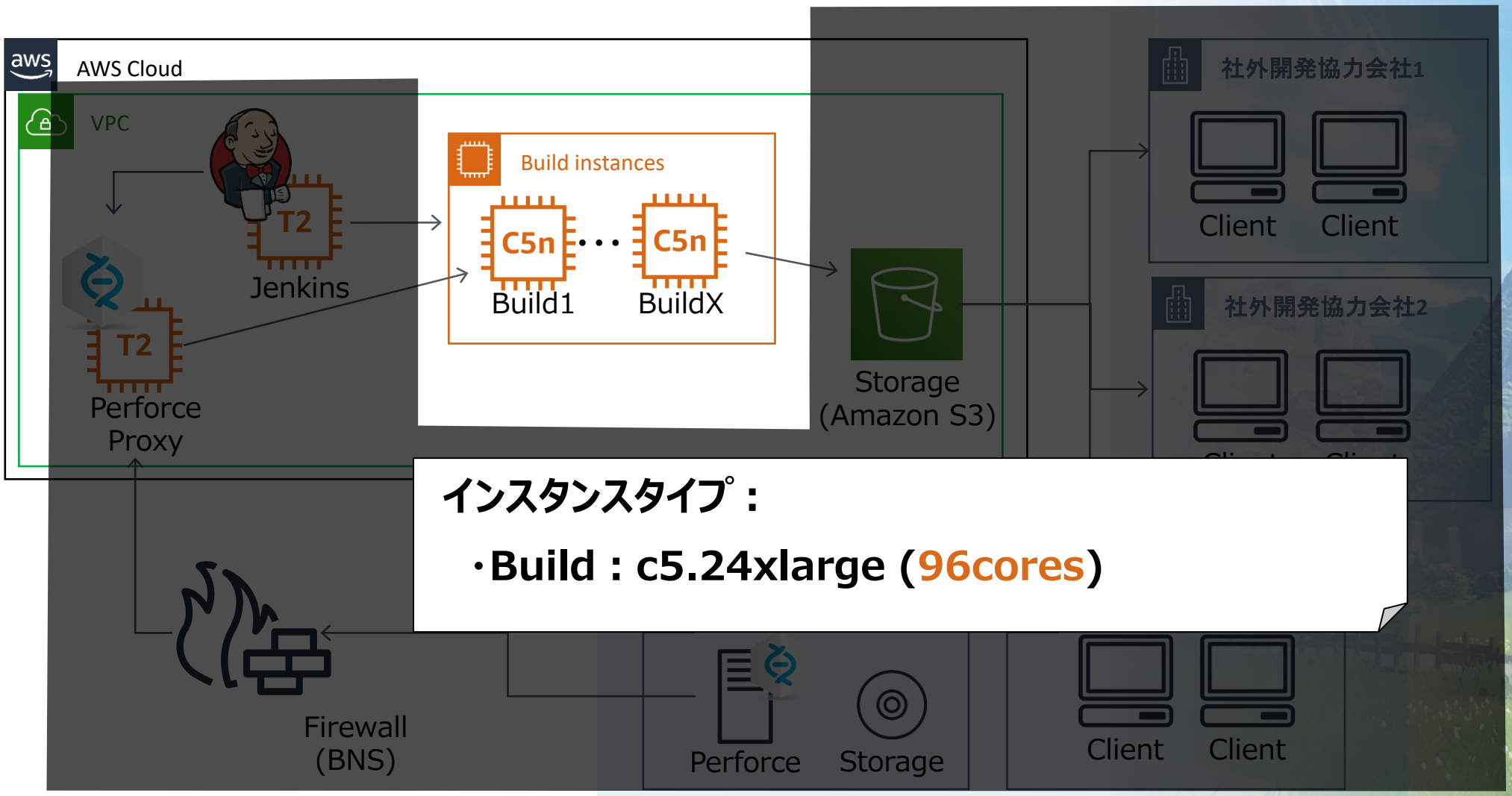
ビルドパイプライン：After

クラウド化したビルド構成



ビルドパイプライン：After

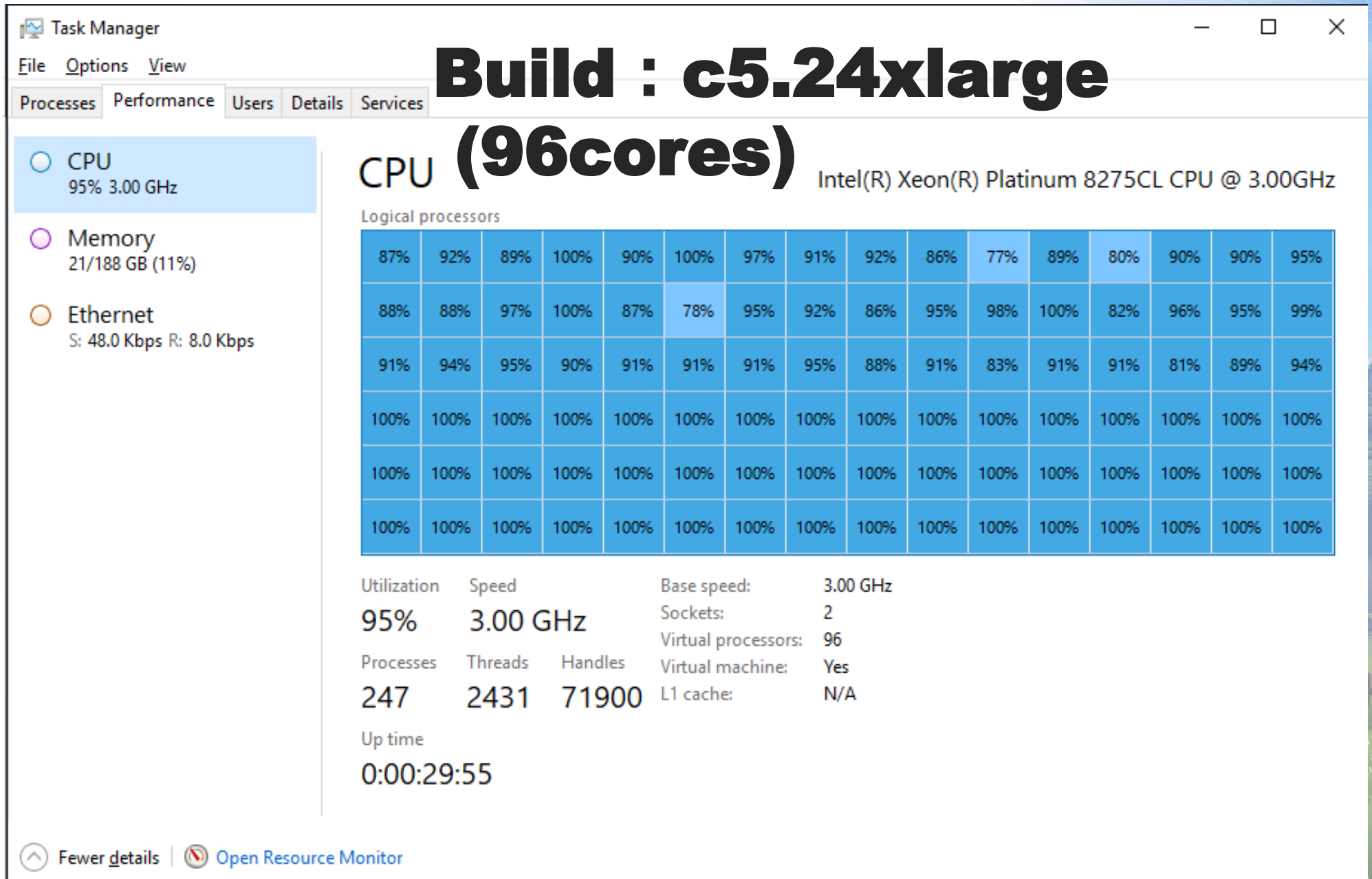
クラウド化したビルド構成



インスタンスタイプ：

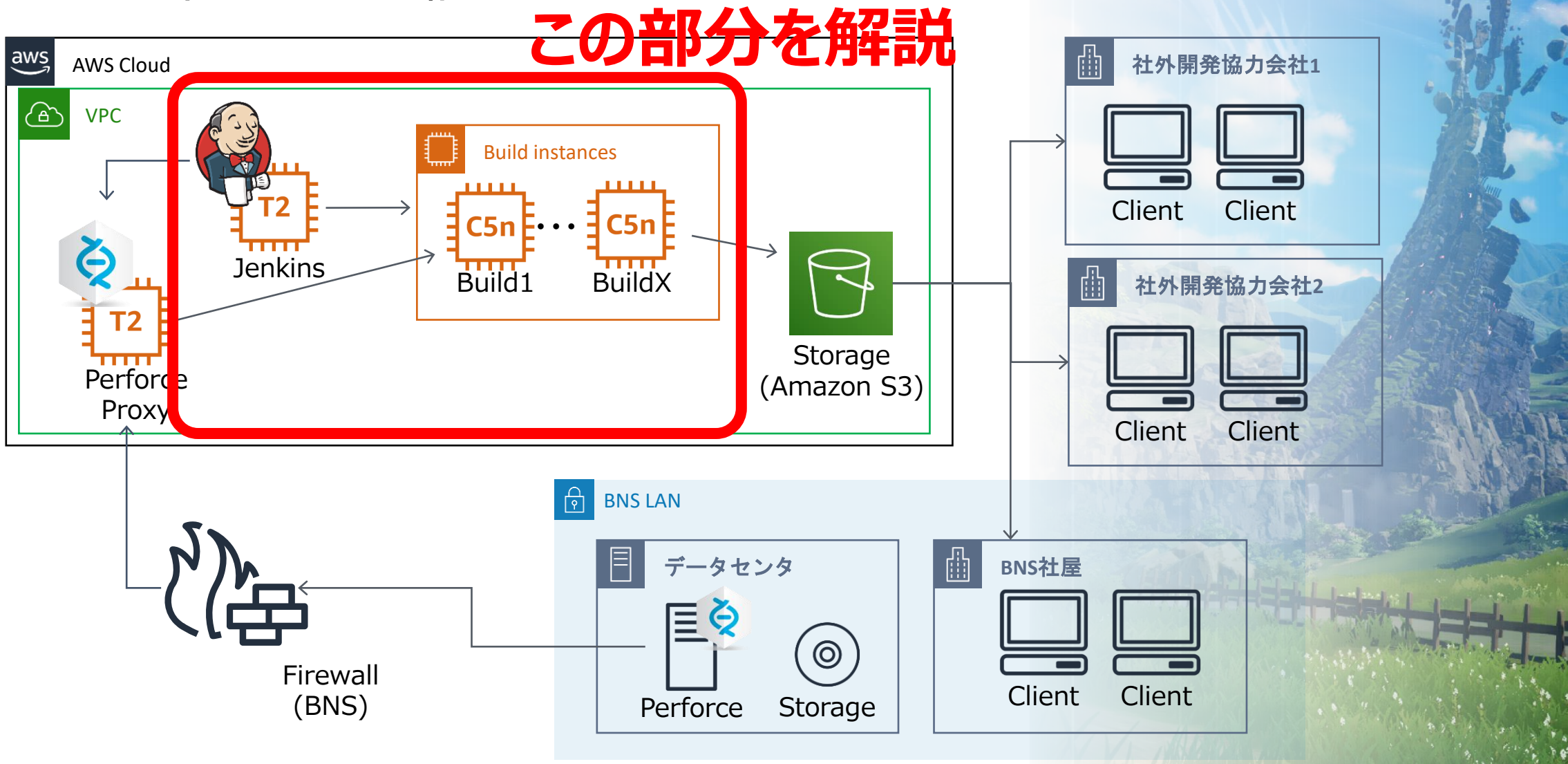
・Build：c5.24xlarge (96cores)

ビルドパイプライン：After



ビルドパイプライン：After

クラウド化したビルド構成



ビルドパイプライン：After

Jenkinsからビルドインスタンスへビルド実行する際の流れ

ビルドインスタンスを起動 →

ビルド開始のキューを送る →

← ビルド結果を返す

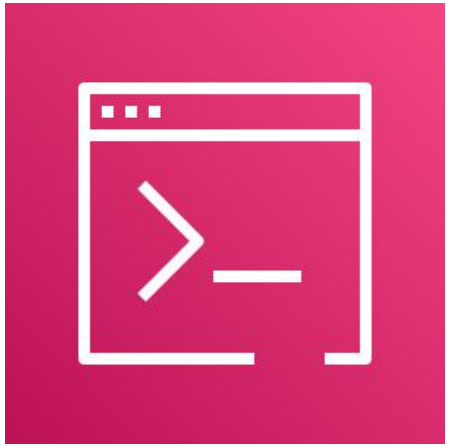
ビルドインスタンスを停止 →

Jenkinsのジョブ解説

利用するツール

Jenkins

ビルドインスタンスの「起動・ビルドリモート実行・ステータスチェック・停止」を管理。Jenkinsインスタンス本体ではビルド実行しない。



AWS CLI

AWSのサービスをコマンドラインから操作・管理するツール。AWSコンソール操作をコマンドラインで操作できる。

AWS Systems Manager

AWSやオンプレミスのリソースを管理するツール。AWS CLIのssmコマンド経由でビルドスクリプトをリモート実行、ステータスや実行結果取得が可能。マネージドインスタンスとして登録する必要がある。

Jenkinsのジョブ解説

インスタンス起動とビルド実行

インスタンス起動

aws ec2 start-instances

--instance-ids **インスタンスID**
--region **リージョン名**

スクリプトのリモート実行

aws ssm send-command

--document-name AWS-RunPowerShellScript
--instance-ids **インスタンスID**
--region **リージョン名**
--parameters commands=**スクリプトのパス** executionTimeout=**制限時間**
--query Command.CommandId

デフォルト実行時間は1時間なので、ビルド実行時間が長い場合、"executionTimeout"を指定する必要がある(ハマった)

AWS System Managerのコマンドが返ってくる

コマンドID

Jenkinsのジョブ解説

```
:STEP1
REM インスタンス起動
for /f "usebackq" %%A in (`aws ec2 describe-instances --instance-id
%instance%  --region ap-northeast-1 --query
Reservations[].Instances[].State.Name --output text`) do set
InstanceState=%%A
if %InstanceState% == stopping aws ec2 wait instance-stopped --
instance-id %instance%  --region ap-northeast-1 --query
Reservations[].Instances[].State.Name --output text
aws ec2 start-instances --instance-ids %instance% --region ap-
northeast-1
aws ec2 wait instance-running --instance-ids %instance% --region ap-
northeast-1
ping -n 30 127.0.0.1
```

Jenkinsのジョブ解説

ビルド状況を確認

ビルド中のステータスを定期的に確認

aws ssm get-command-invocation

--command-id **コマンドID**
--instance-id **インスタンスID**
--region **リージョン名**
--output text
--query Status

Inprogress

コマンドIDのステータスが返ってくる

コマンドステータス

ビルド成功/失敗確認へ

Jenkinsのジョブ解説

```
:STEP2
REM ビルドコマンドを実行。終了までステータスチェックを繰り返す
aws ssm send-command --document-name AWS-RunPowerShellScript --
instance-ids %instance% --region ap-northeast-1 --parameters
commands=%command% executionTimeout=18000 --query Command.CommandId --
output text > tmp.txt
type tmp.txt
for /f "usebackq" %%A in (`type tmp.txt`) do set CommandId=%%A
del tmp.txt

:DoCmdStatChk
for /f "usebackq" %%A in (`aws ssm get-command-invocation --command-id
%CommandId% --instance-id %instance% --region ap-northeast-1 --output
text --query Status`) do set CommandStatus=%%A
ping -n 1 127.0.0.1
if %CommandStatus% == InProgress goto DoCmdStatChk
```

Jenkinsのジョブ解説

ビルド成功/失敗の判定とインスタンス停止

ビルドスクリプトの標準エラー出力確認

aws ssm get-command-invocation

--command-id **コマンドID**
--instance-id **インスタンスID**
--region **リージョン名**
--output text
--query StandardErrorContent

C5n

標準エラー出力結果が返ってくる

StandardErrorContent

インスタンス停止

aws ec2 stopt-instances

--instance-ids **インスタンスID**
--region **リージョン名**

Next ビルド結果に応じた処理を進める(e.g. Slackへ通知)

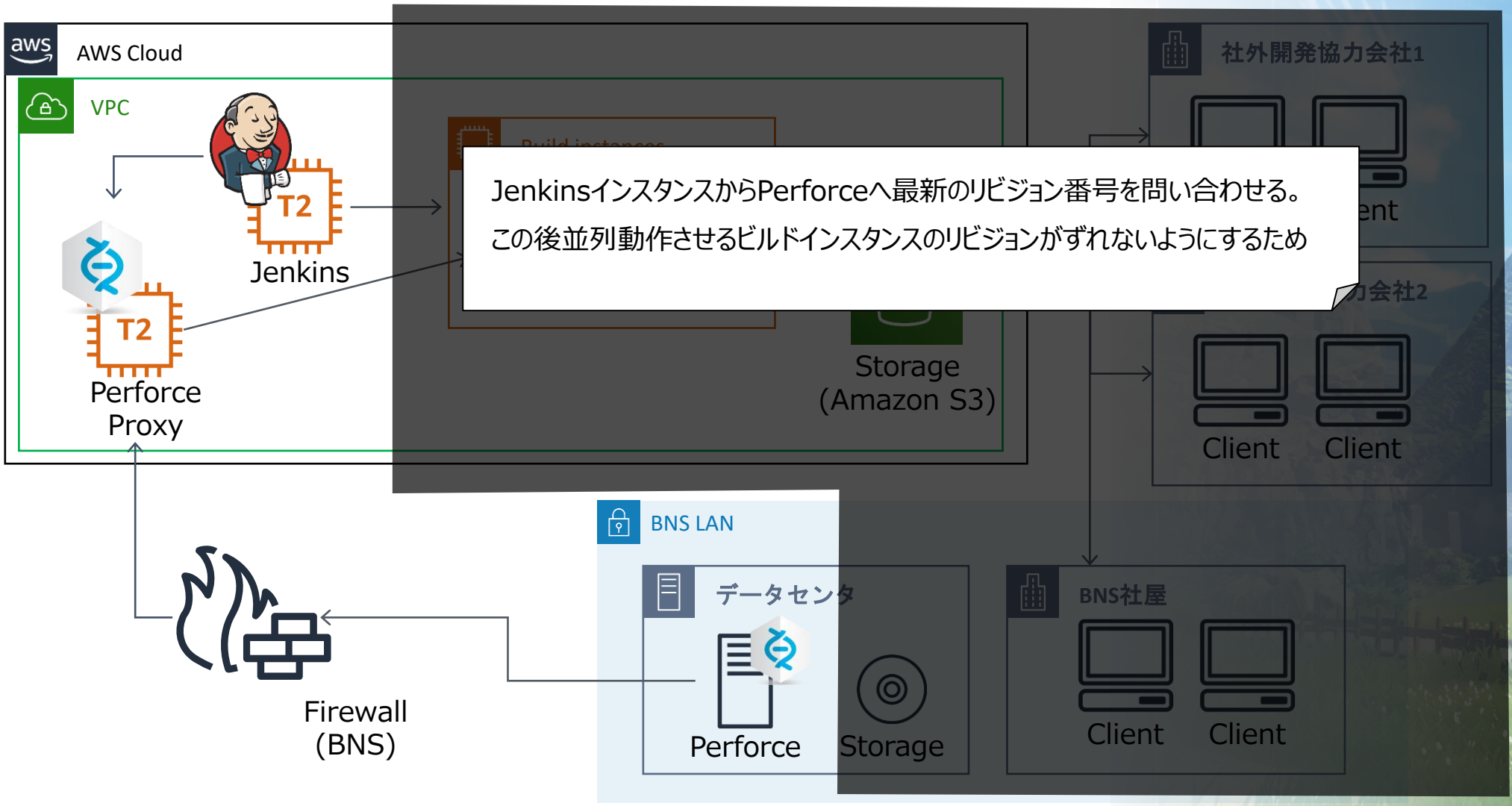
Jenkinsのジョブ解説

```
:STEP3
REM コマンド実行結果からビルド結果を判定。インスタンス終了
aws ssm get-command-invocation --command-id %CommandId% --instance-id
%instance% --region ap-northeast-1 --output text --query StandardOutputContent
> CmdSOC.log
aws ssm get-command-invocation --command-id %CommandId% --instance-id
%instance% --region ap-northeast-1 --output text --query StandardErrorContent >
CmdSEC.log
for /f "usebackq" %%A in (`findstr "." CmdSOC.log`) do set CmdSOC=%%A
for /f "usebackq" %%A in (`findstr "." CmdSEC.log`) do set CmdSEC=%%A

if %CmdSEC%_nul == _nul (
  goto JobSuccess
 ) else (
   goto JobFailed
 )
:JobSuccess
echo BUILD SUCCESS
aws ec2 stop-instances --instance-ids %instance% --region ap-northeast-1
exit 0
:JobFailed
echo BUILD FAILED
aws ec2 stop-instances --instance-ids %instance% --region ap-northeast-1
exit 1
```

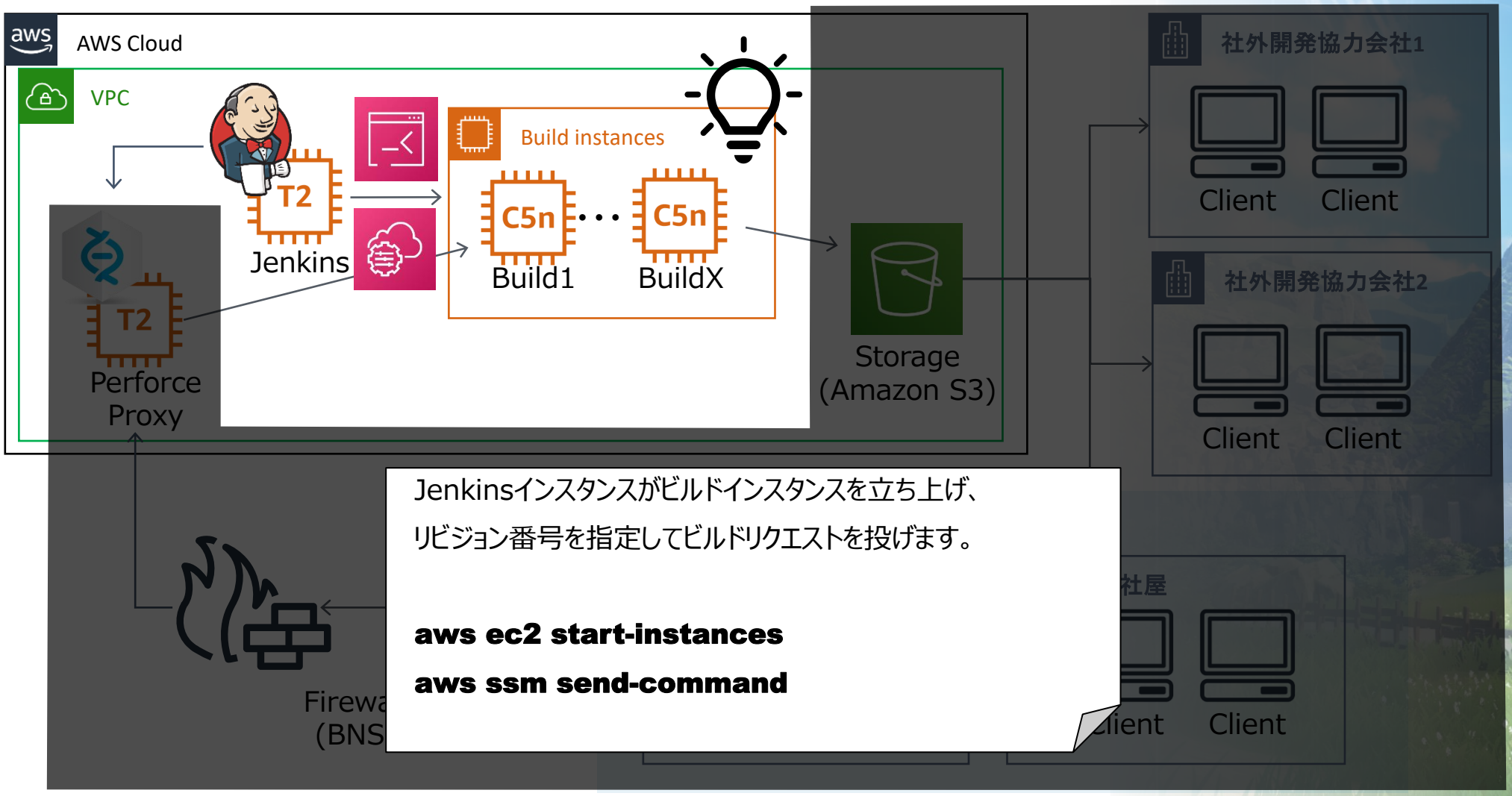
ビルドパイプライン：After

クラウド化したビルドの流れ



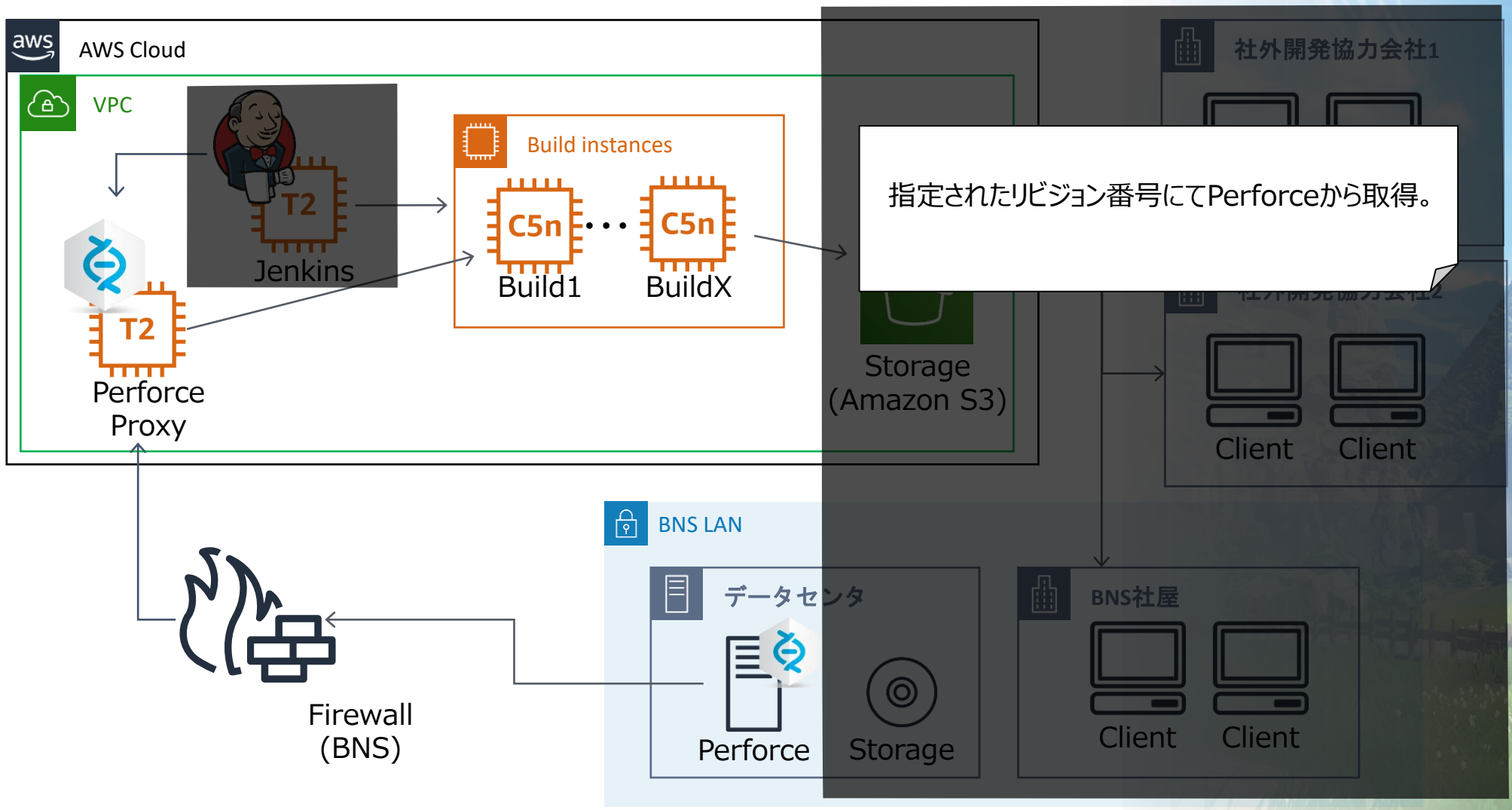
ビルドパイプライン：After

クラウド化したビルドの流れ



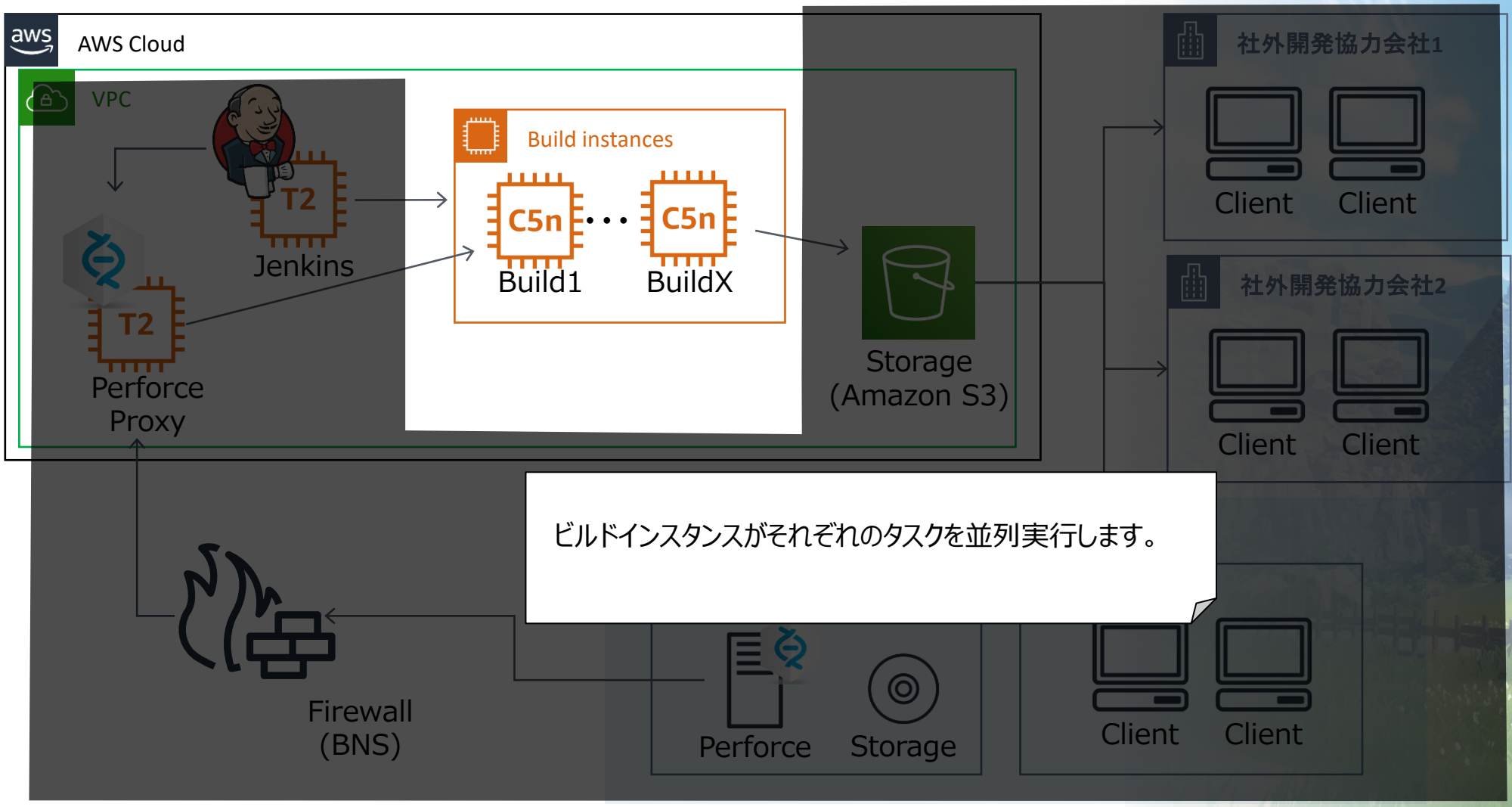
ビルドパイプライン：After

クラウド化したビルドの流れ



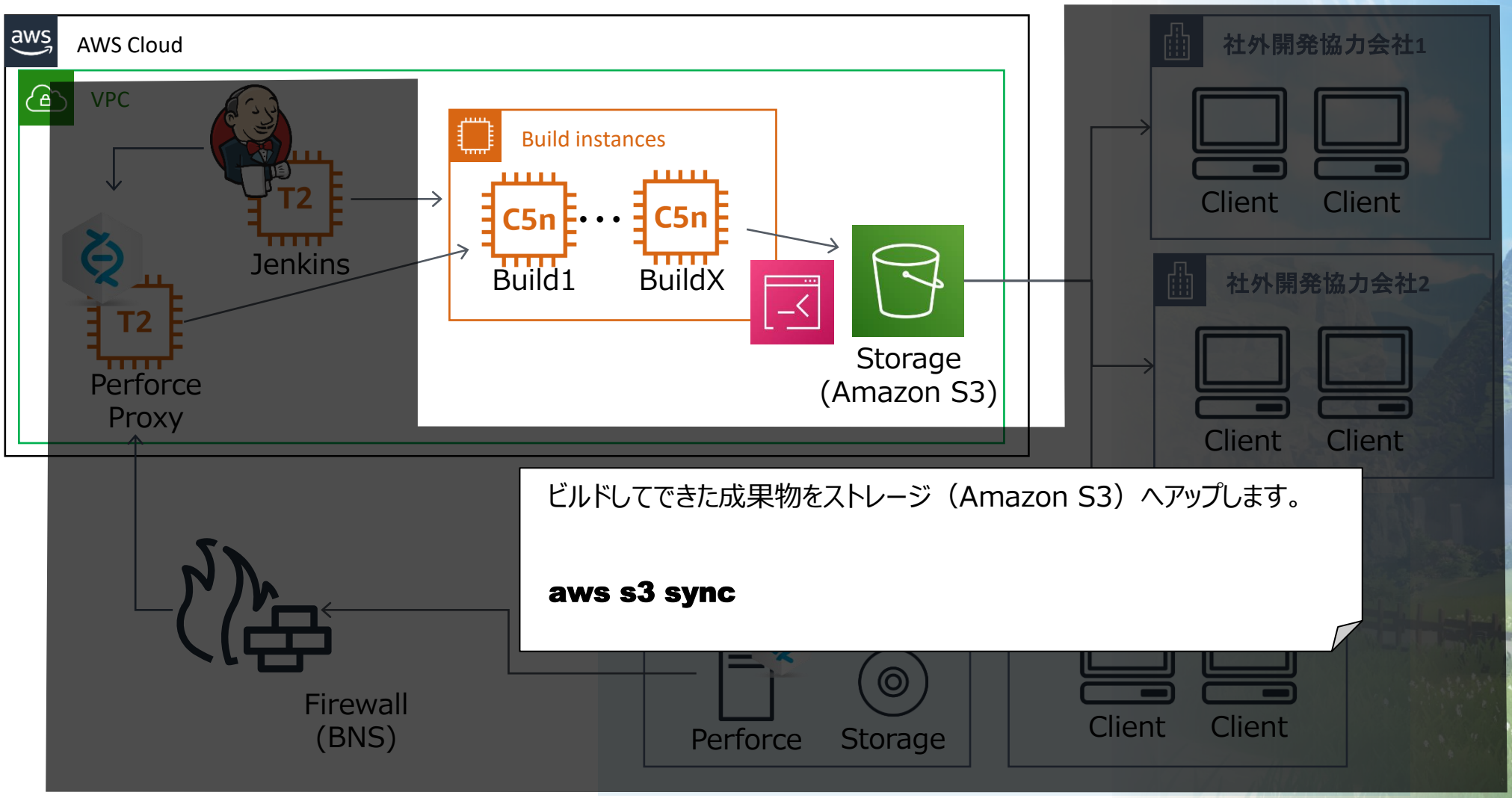
ビルドパイプライン：After

クラウド化したビルドの流れ



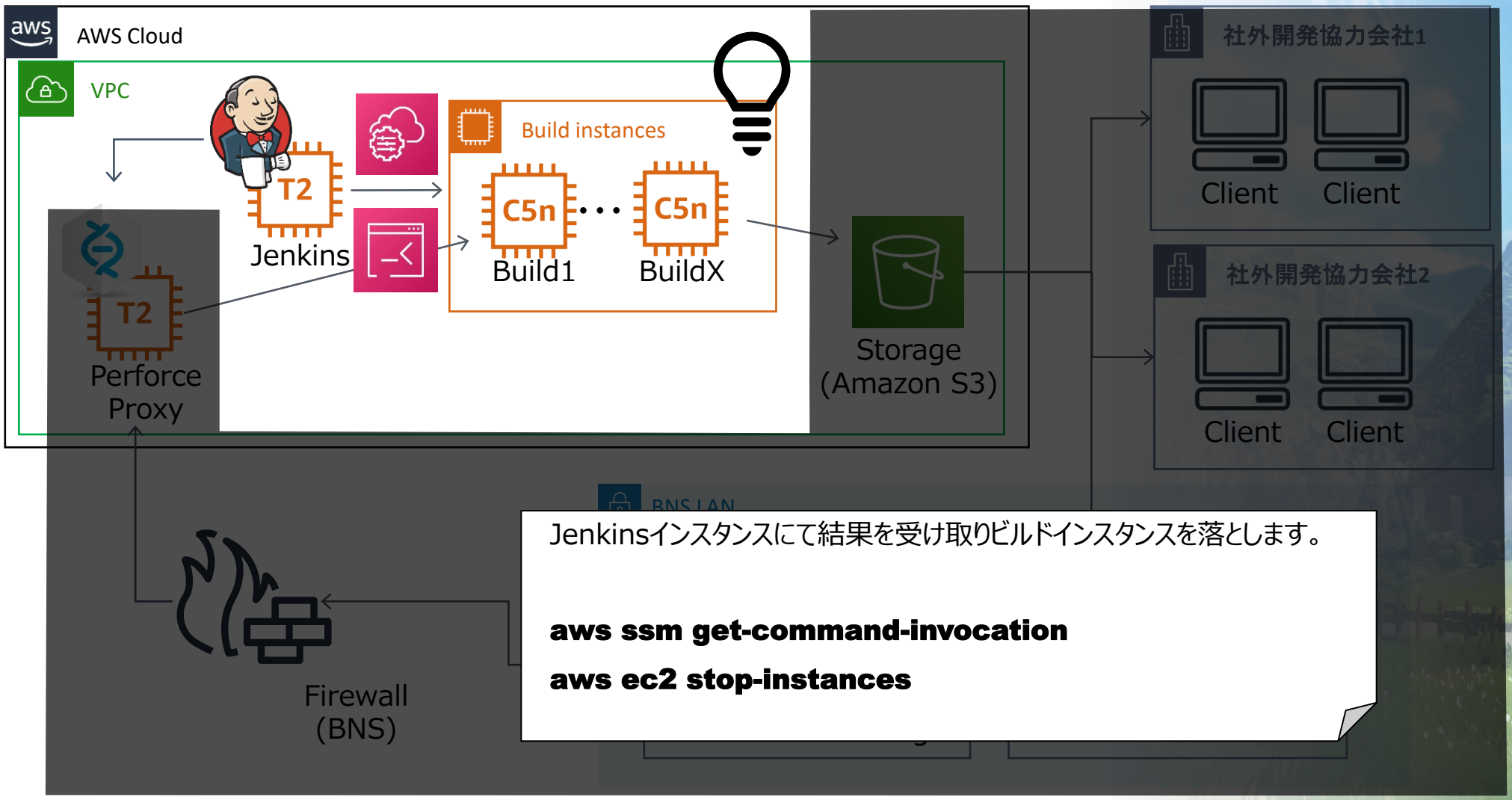
ビルドパイプライン：After

クラウド化したビルドの流れ



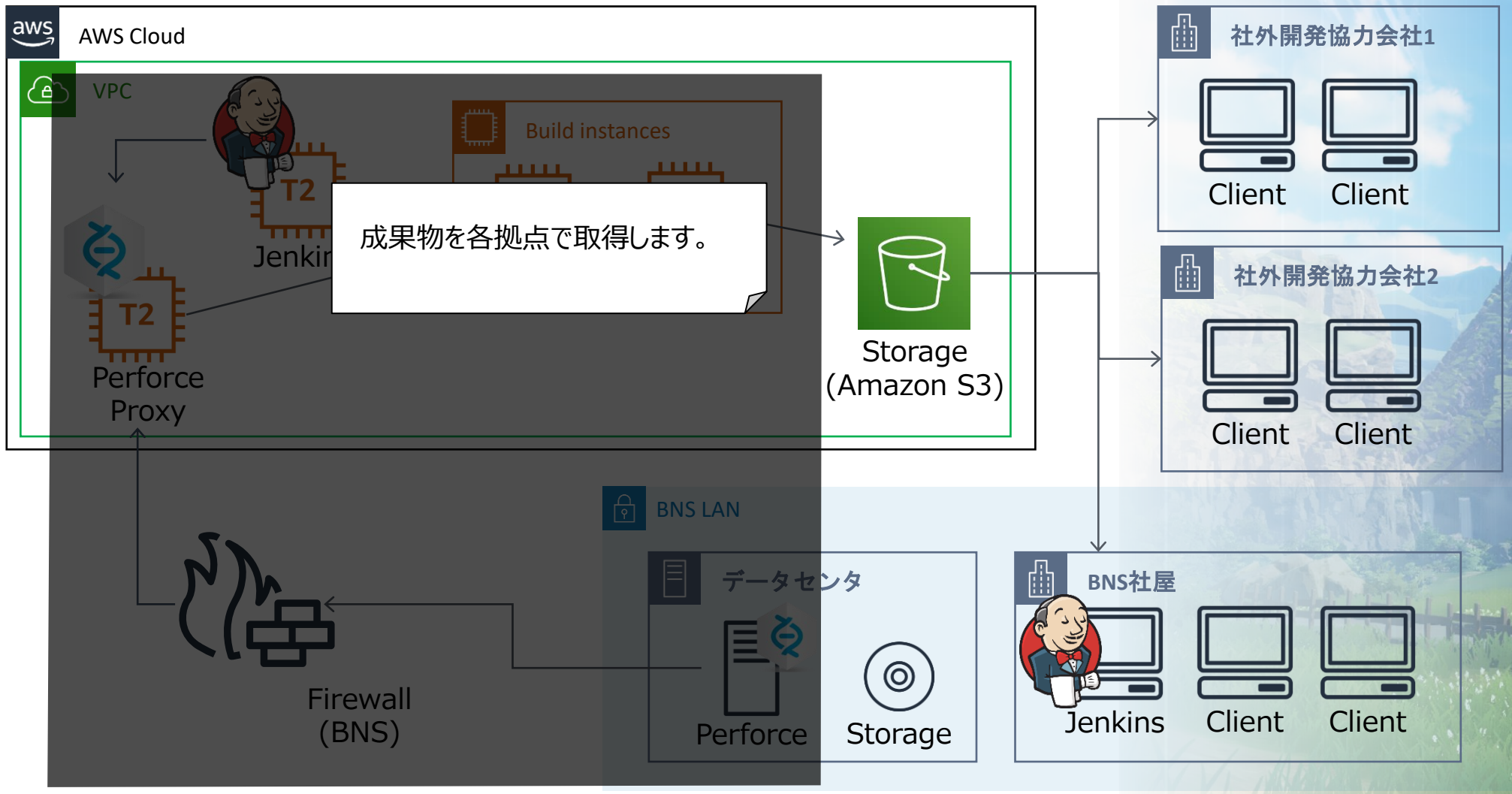
ビルドパイプライン：After

クラウド化したビルドの流れ



ビルドパイプライン：After

クラウド化したビルドの流れ



アジェンダ

1. BLUE PROTOCOLとは？
2. ゲーム開発でのビルドパイプライン解説
3. BLUE PROTOCOLのビルドパイプラインクラウド化した話
4. **クラウド化した結果や成果、わかった事**
5. １年半運用した現状
6. 運用中に改善した事

クラウド化した結果や成果、わかった事

結果・成果・わかった事

150コア！

	ローカルPC	ローカルPC並列化 (IncrediBuild)	クラウド (c5.24xlarge)
Compile	約96分	約5分	約18分
Cook	約80分	約80分	約83分
Packaging	約62分	約66分	約25分
合計	約3.9時間	約2.5時間	約2.1時間

キャッシュ次第なので、
あまり参考にならない

コア数が多いほうが早い

IncrediBuildがなくてもクラウドで遜色ないくらいの速度は出る

クラウド化した結果や成果、わかった事

ランニングコスト

サービス	項目	インスタンスタイプ	単価/時(USD)	起動時間/日(h)	金額/30日(USD)
EC2	Jenkins (Windows)	t2.micro	0.0198	24	14.26
EC2	Build1(C5 Windows)	c5d.24xLarge	10.272	2	616.32
EC2	Build2(C5 Windows)	c5d.24xLarge	10.272	1.5	462.24
EC2	Build3(G4 Windows)	g4dn.16xlarge	8.819	1	264.57
EC2	P4P(Linux)	t2.small	0.0396	24	28.51

サービス	項目	ストレージタイプ	容量(GB)	利用時間/日	金額/30日
EBS	Build1のストレージ	gp2	1000	2	9.86
EBS	Build2のストレージ	gp2	1000	1.5	7.40
EBS	Build3のストレージ	gp2	1000	1	4.93
EBS	P4Pのストレージ	gp2	2000	24	236.71
EBS	Jenkinsのストレージ	gp2	30	24	3.55

サービス	項目	一回のサイズ(GB)	POSTリクエスト回数/日	GETリクエスト数/日	金額/30日
S3	成果物のアップロード先	10	1	10	7.50

合計：
1655.85 USD/月

構成：
- P4Proxy x1
- Jenkins x1
- Build x2
- Build(Bake) x1
- Strage(S3) x1

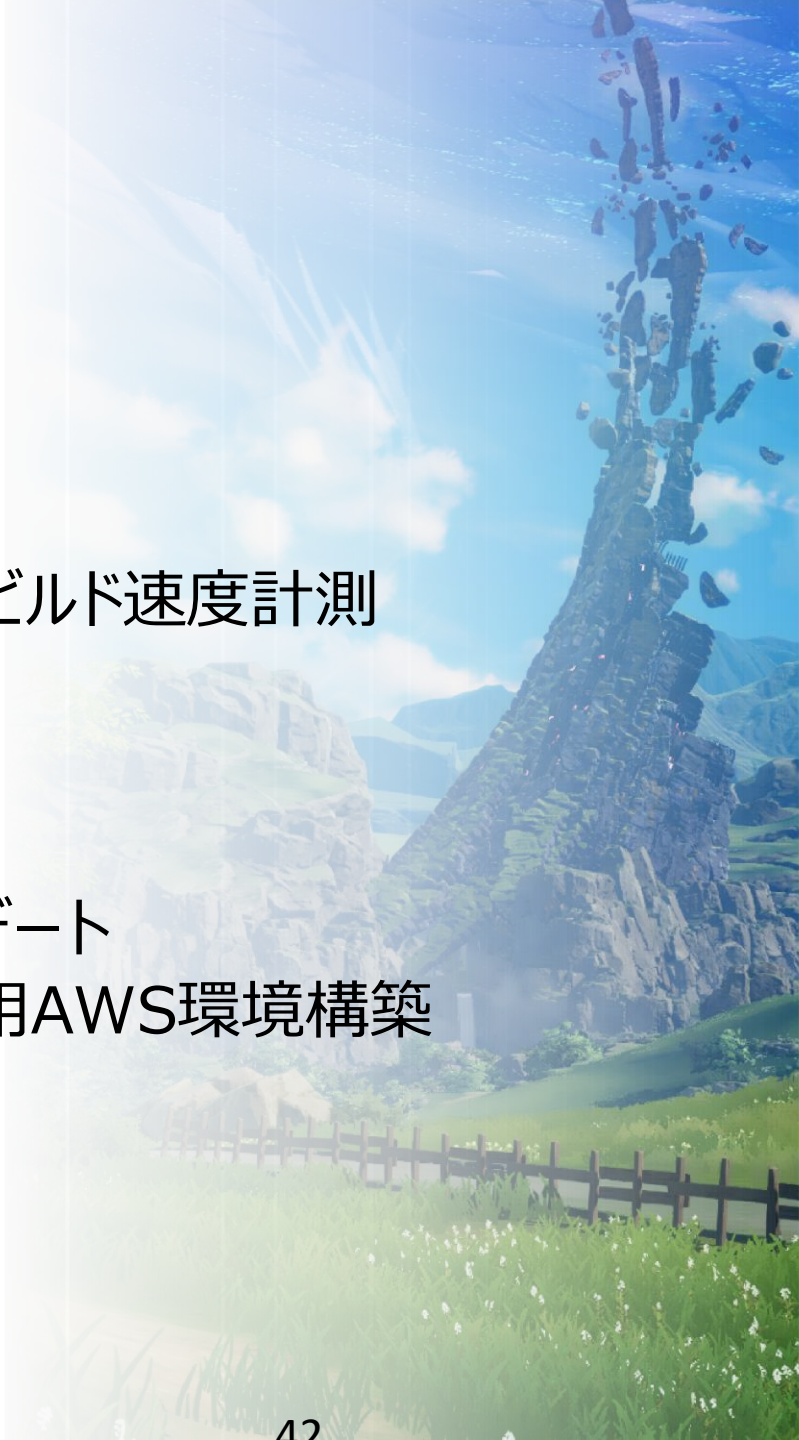
リージョン：
・アジアパシフィック
(東京) ap-northeast-1

クラウド化した結果や成果、わかった事

検証から導入までのスケジュール感

2019年
- 11月　検証スタート
- 12月　AWS環境構築・ Jenkins無しでP4P経由でビルド速度計測

2020年
- 1月　ビルドパイプライン構成を決定
- 2月　Jenkinsジョブの作りこみ、ビルドスクリプトアップデート
- 5月　CloudFormationによるBLUE PROTOCOL用AWS環境構築
- 6月　プロジェクトでの実運用開始



クラウド化した結果や成果、わかった事

AWS良かった事・悪かった事（構築してみた所感）

■ GOOD

・マシン調達の待ち時間：**数週間→数分**
・ビルド性能：**社内の分散環境とほぼ同等**
・アクセスコントロール：**取引先とのデータ授受等も柔軟**
・ネット上のナレッジ：**公式だけでなくユーザー記事も多い**

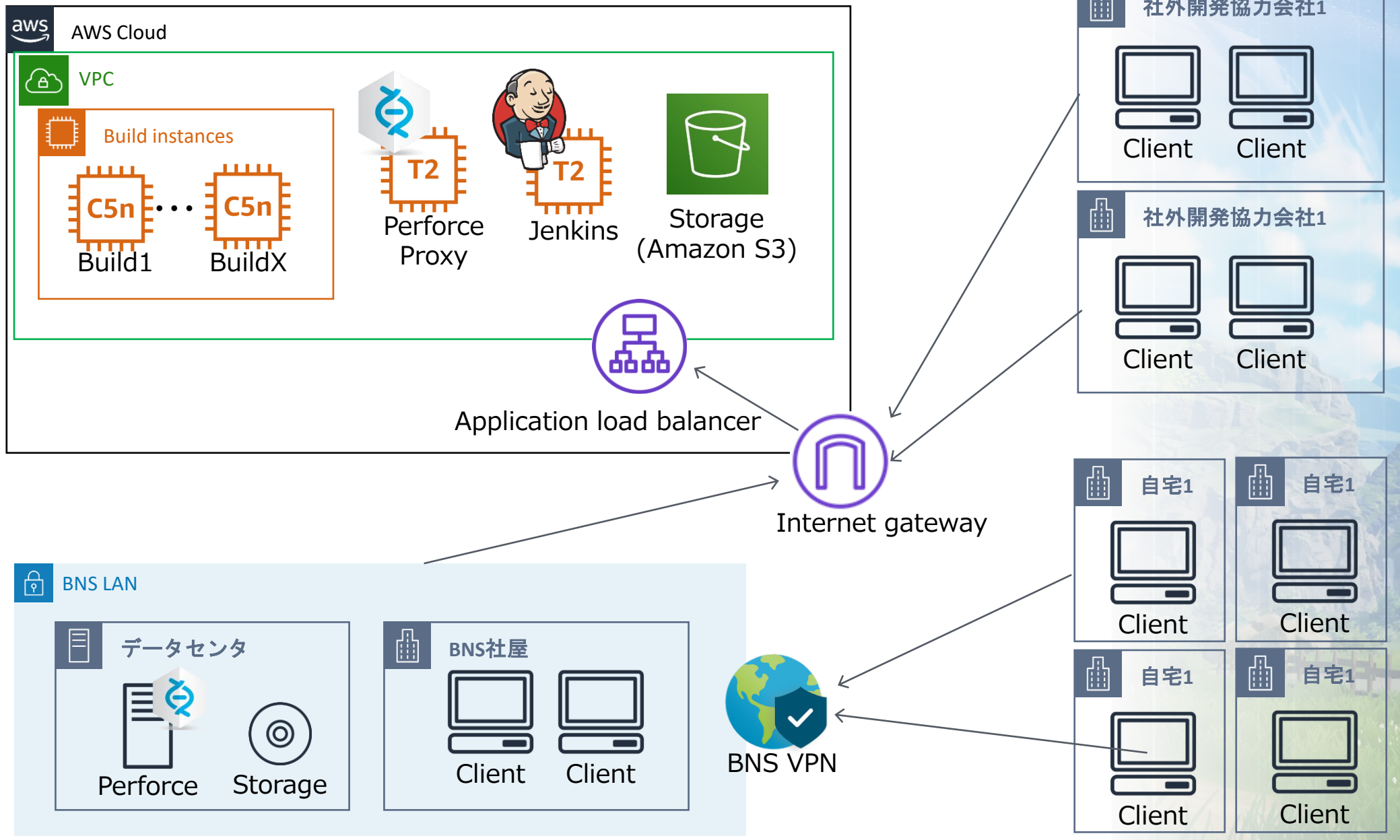
■ BAD

・従量課金制：**稼働時間が長いと金額が怖い**
・Visual Studio：**使えない(ライセンス問題)**
・技術のキャッチアップが大変：**サービスの追加更新が早い(うれしい悲鳴)**
・コスト試算：**複数サービス使うと試算が大変**

アジェンダ

1. BLUE PROTOCOLとは？
2. ゲーム開発でのビルドパイプライン解説
3. BLUE PROTOCOLのビルドパイプラインクラウド化した話
4. クラウド化した結果や成果、わかった事
5. **１年半運用した現状**
6. 運用中に改善した事

在宅リモートワークとビルド構成



1年半運用した現状

ビルド時間の増大

	クラウド移行直後 (c5.24xlarge)	現状 (c5.24xlarge)
Compile	約18分	約20分
Cook	約83分	約154分
Packaging	約25分	約26分
合計	約2.1時間	約3.3時間

コードが増えたことによる
コンパイル時間の増大は軽微

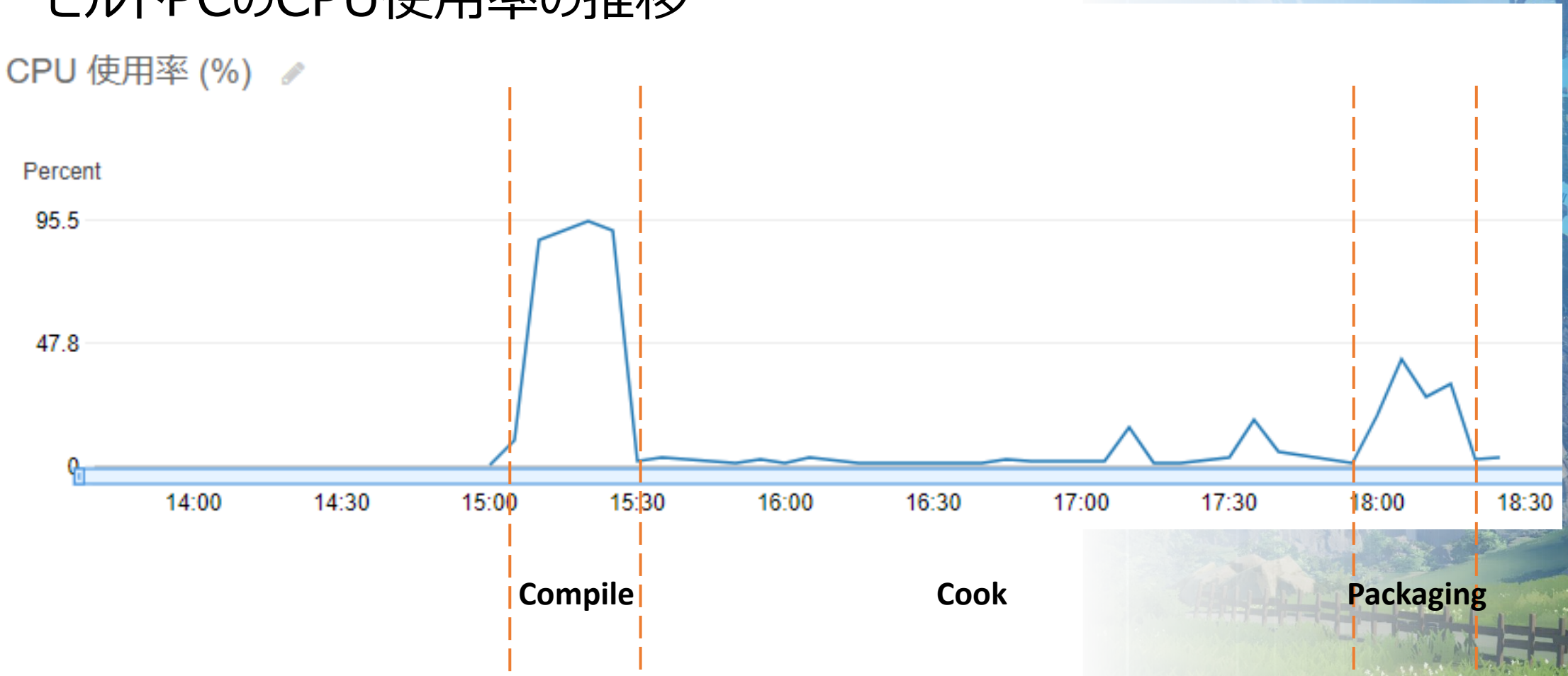
アセットが増えたことによる
クック時間の増大は甚大

パックの分割数も増えたので
並列化されて問題ない

複数コアで並列化できないCookが問題となる

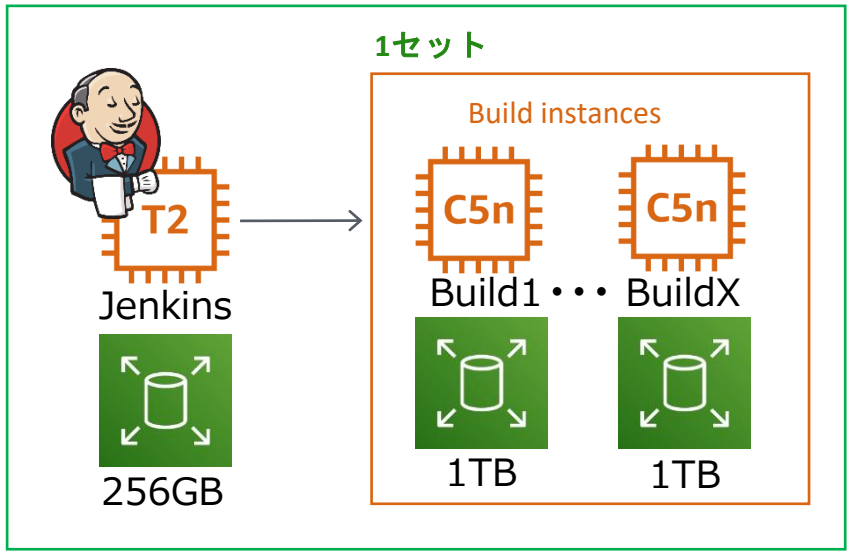
1年半運用した現状

ビルドPCのCPU使用率の推移



1年半運用した現状

(JenkinsPC + ビルドPC x 2) x 10環境
ビルドPC自体は必要に応じて起動するが、
ストレージ(EBS)が馬鹿にならない



Jenkins常時稼働	約8,600時間	約$600
ビルド適時稼働	約640時間	約$6,700
ストレージ	約25TB	約$3,000

約$10,000/月

1年半運用した現状

立ちふさがった壁

1. ビルドインスタンスの進捗が把握しづらい
2. クラウドと社内のJenkinsの連携
3. 在宅リモートワークによるダウンロード時間
4. インスタンスへのRDP
5. 停止中インスタンスのEBS
6. 複数環境でのJenkinsアカウント管理
7. ハイスペックインスタンスの動作していないCPU

アジェンダ

1. BLUE PROTOCOLとは？
2. ゲーム開発でのビルドパイプライン解説
3. BLUE PROTOCOLのビルドパイプラインクラウド化した話
4. クラウド化した結果や成果、わかった事
5. １年半運用した現状
6. **運用中に改善した事**

運用中に改善した事

立ちふさがった壁

1. **ビルドインスタンスの進捗が把握しづらい**
2. クラウドと社内のJenkinsの連携
3. 在宅リモートワークによるダウンロード時間
4. インスタンスへのRDP
5. 停止中インスタンスのEBS
6. 複数環境でのJenkinsアカウント管理
7. ハイスペックインスタンスの動作していないCPU

ビルドPCのJenkinsスレーブ化

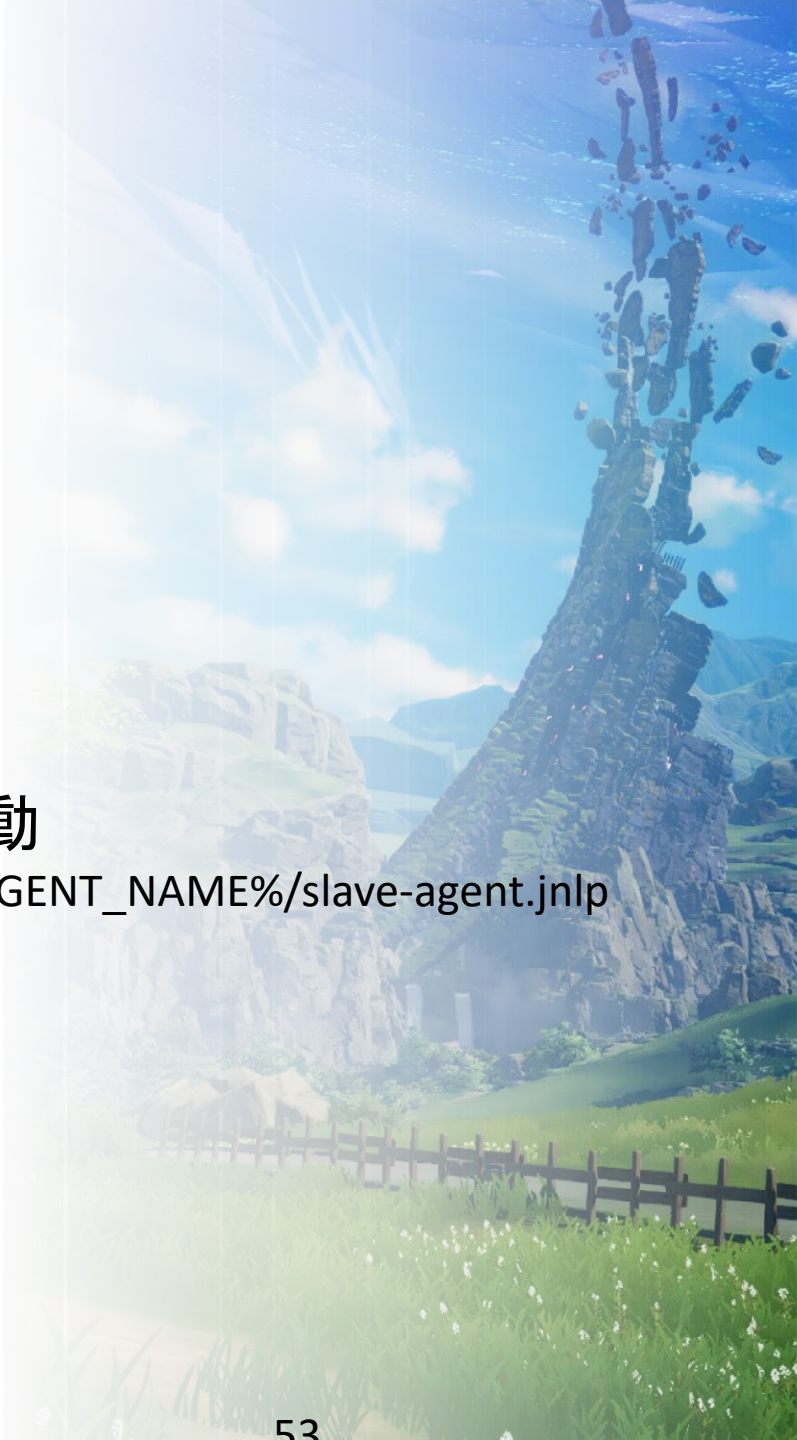
- JenkinsPCとビルドPCとのやり取りをSSMだけだと限界
  - send-commandの結果（stdout）

- Jenkinsにスレーブエージェントを使えばいいのでは？

ビルドPCのJenkinsスレーブ化

ビルドPCの設定

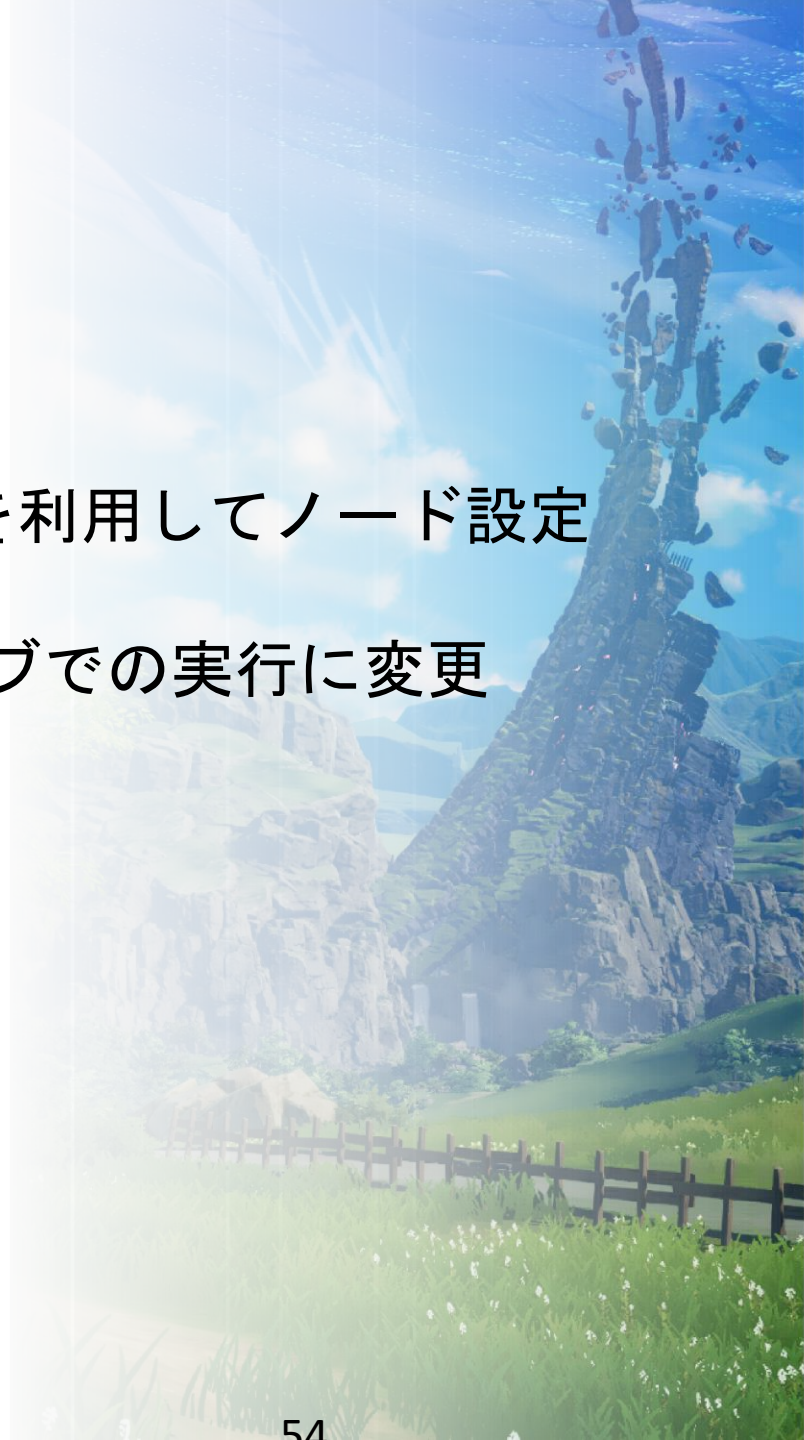
- Amazon Corretto 8のインストール
  - 無償のマルチプラットフォーム対応のJDK
- Jenkinsからagent.jarをダウンロードして配置
- タスクスケジューラで起動時にエージェント起動
  - java.exe -jar c:¥agent.jar -jnlpUrl %AGENT_HOST%/computer/%AGENT_NAME%/slave-agent.jnlp -secret %AGENT_SECRET%



ビルドPCのJenkinsスレーブ化

JenkinsPCの設定

- WMI Windows Agentsプラグインを利用
  - 方法はいくつかあるがお手軽なプラグインを利用してノード設定
- aws ssm send-commandしていたところをスレーブでの実行に変更



ビルドPCのJenkinsスレーブ化

結果の確認がしやすくなった

- スレーブの設定は少々手間
- ビルドPCで実行された結果（stdout）がすべて確認できるようになったので、問題が起きた時なども対処がしやすい

運用中に改善した事

立ちふさがった壁

1. ビルドインスタンスの進捗が把握しづらい
2. **クラウドと社内のJenkins連携**
3. 在宅リモートワークによるダウンロード時間
4. インスタンスへの停止中インスタンスのEBS
5. 複数環境でのJenkinsアカウント管理
6. ハイスペックインスタンスの動作していないCPU

Slackを介したやり取り

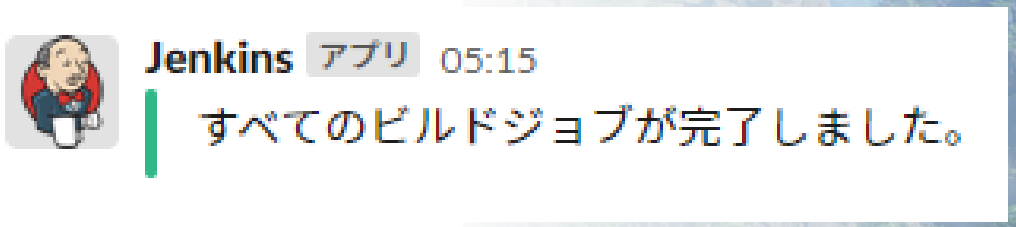
クラウドJenkinsが終了したら、社内Jenkinsが働く

- 定期的に実行だけでは不十分
- お手軽な方法がないものか？

Slackを介したやり取り

クラウドJenkinsからSlackへの通知

- Jenkinsプラグイン「Slack Notification」
- Slackアプリ「Jenkins CI」



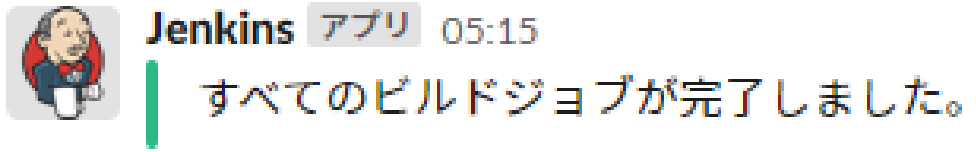
Slackを介したやり取り

Slackへの通知を検知し、社内Jenkinsをトリガー

- Slackアプリ「Bots」
- Slackbot for Python

Slackを介したやり取り

Slackへの通知を検知し、社内Jenkinsをトリガー




```
@listen_to('すべてのビルドジョブが完了しました。')
def listen_func( message ):
    api = 'http://jenkins-pc:8080/job/copy_package/build'
    requests.post( api, auth = None)
```

- @listen_to('すべてのビルドジョブが完了しました。')デコレータにて投稿を検知
- Requestsライブラリのpostにて、JenkinsAPIを使って「copy_package」ジョブを叩く！

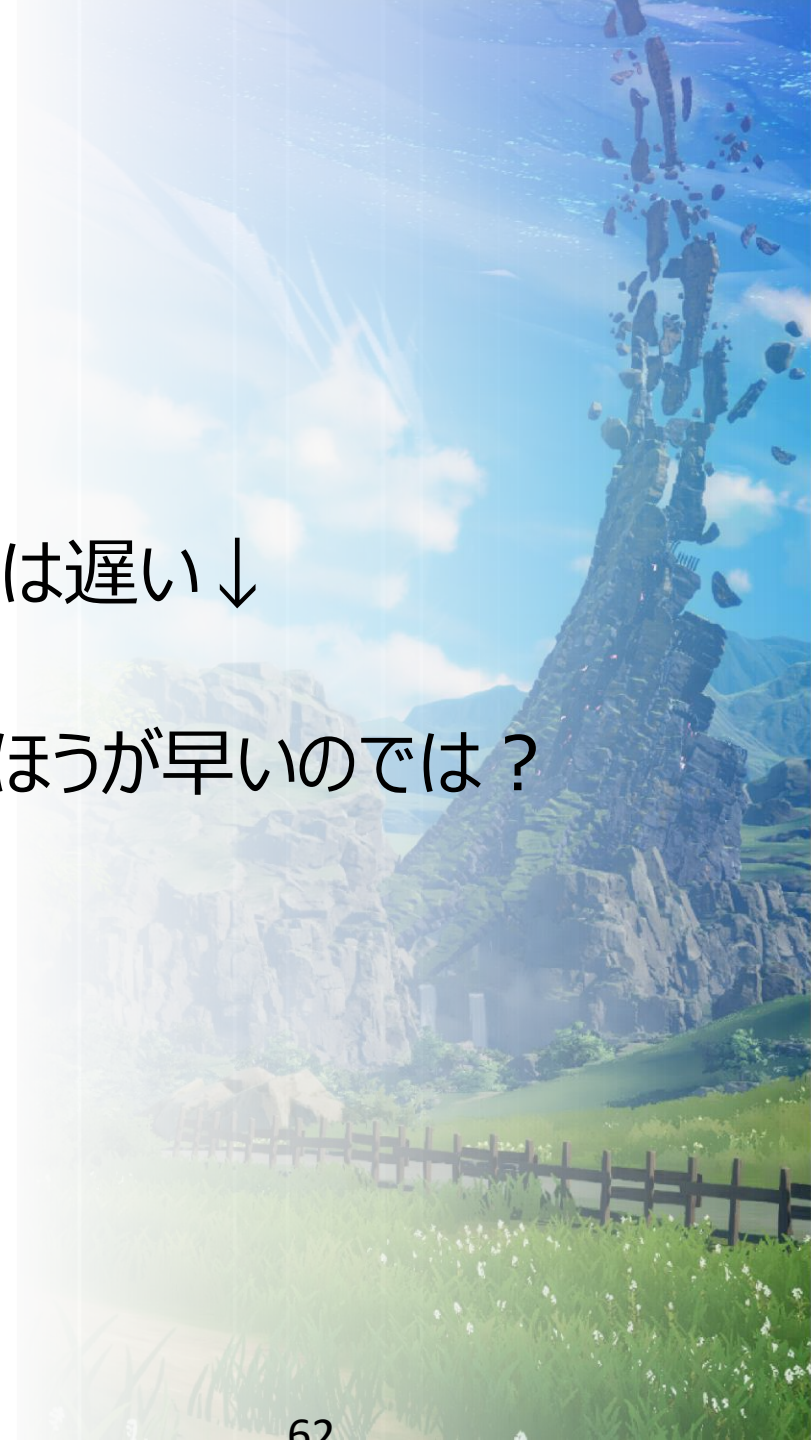
運用中に改善した事

立ちふさがった壁

1. ビルドインスタンスの進捗が把握しづらい
2. クラウドと社内のJenkins連携
3. **在宅リモートワークによるダウンロード時間**
4. インスタンスへのリモートデスクトップ
5. 停止中インスタンスのEBS
6. 複数環境でのJenkinsアカウント管理
7. ハイスペックインスタンスの動作していないCPU

成果物を直接クラウドからダウンロード

- 在宅リモートワークが急増！
- ファイルサーバーからのVPN経由のダウンロードは遅い↓
- せっかくクラウドなのだから直接ダウンロードしたほうが早いのでは？



成果物を直接クラウドからダウンロード

セキュリティの担保

AWS Identity and Access Management（IAM）

1. アクセスキーにて認証
   - 定期的に更新

2. 成果物の格納されているディレクトリのみへのアクセス
   - Bucket/Path/

3. 読み込みのみ許可（ReadOnly）
   - リストも禁止

成果物を直接クラウドからダウンロード

セキュリティの担保

- ファイルリストを社内ファイルサーバーにて管理
  - VPN経由で接続
- パスとハッシュの組み合わせ

```
Game.exe, 3729bb51fa1ca21b40c0c8ca50d292b6647f42e0
Game¥Binaries¥Win64¥Game.exe, 29e6feff9fb16b9d8271b7da6925baf3c6339d06
Game¥Content¥Paks¥pakchunk0-WindowsClient.pak,68d96b0f391259702765cd5b7169d27196bb3d1e
Game¥Content¥Paks¥pakchunk1-WindowsClient.pak,085efbc542a7d5314c133d1eb7a1cae0174d53f3
 :
 :
```

成果物を直接クラウドからダウンロード

セキュリティの担保

AWS SDK for Python（Boto3）

- Pythonにてダウンロードツールを作成
  - ユーザーはAWS CLIをインストールしなくてもよい
- ファイルリストをもとにダウンロード
  - 更新時はローカルファイルとハッシュを比較
- PyInstallerにてexe化
  - ユーザーはPythonも入れる必要がない
  - pyinstaller downloader.py --onefile

成果物を直接クラウドからダウンロード

```
# セッション接続
session = Session(aws_access_key_id=ACCESS_KEY, aws_secret_access_key=SECRET_KEY, region_name=REGION)
bucket = session.resource('s3').Bucket(BUCKET_NAME)

for filename in filenames:
        str = filename.split(',')
        path = str[0]
        server_hash = str[1]

        # パス
        file_path = download_dir + path
        bucket_url = bucket_dir + path.replace('¥¥', '/')

        # ハッシュ確認
        local_hash = ""
        if (os.path.exists(file_path)):
            with open(file_path, 'rb') as f:
                data = f.read()
                local_hash = hashlib.sha1(data).hexdigest()

        if server_hash == local_hash:
            continue

        # ダウンロード
        bucket.download_file(bucket_url, file_path)
```

成果物を直接クラウドからダウンロード

ダウンロード高速化に成功！

- ファイルリストをファイルサーバーに置いているためVPN接続が必要

- S3へのアクセスをVPNスルーする
  FQDN ：バケット名.s3.ap-northeast-1.amazonaws.com
  ポート：443

- 家庭環境の最高速！
  - 約15分 ⇒ 約2分

- データ転送にかかる費用は軽微
  - $0.2未満/GB
  - 5GB/１回

⇒毎日1人1ドルくらい

運用中に改善した事

立ちふさがった壁

1. ビルドインスタンスの進捗が把握しづらい
2. クラウドと社内のJenkins連携
3. 在宅リモートワークによるダウンロード時間
4. **インスタンスへのリモートデスクトップ**
5. 停止中インスタンスのEBS
6. 複数環境でのJenkinsアカウント管理
7. ハイスペックインスタンスの動作していないCPU

VPC内のWindowsインスタンスへリモートデスクトップ

インスタンスを作成後、すぐにリモートデスクトップしたい

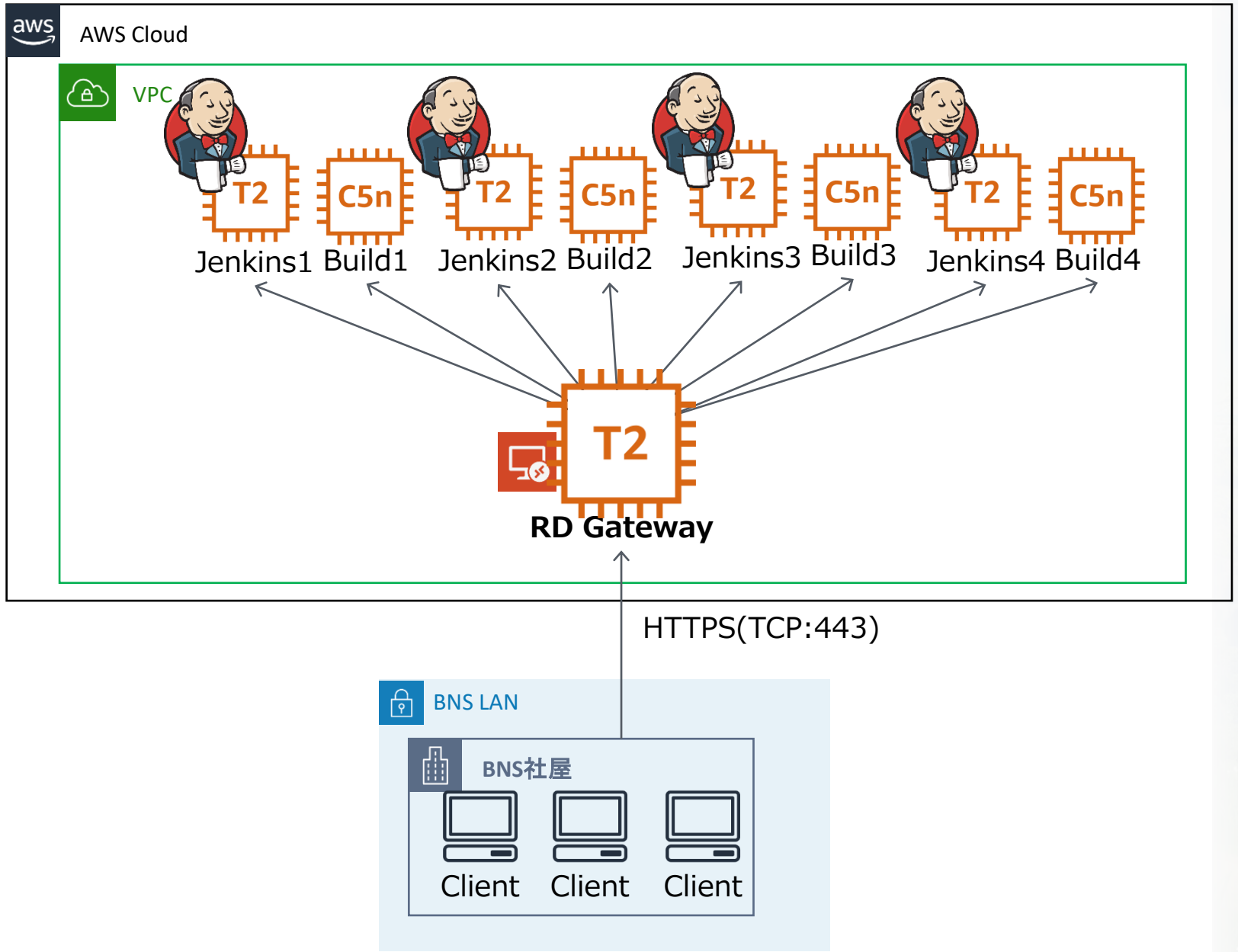
↓

RD Gatewayインスタンスを導入

RD Gatewayとは？

- RDPの代わりにHTTPSでインスタンスへ接続する
- SSL通信なのでRDPよりもセキュア
- EIP(Elastic IP)を持たないインスタンスにも接続できる
- ファイアーウォール許可申請はRD Gatewayインスタンスだけ

VPC内のWindowsインスタンスへリモートデスクトップ



運用中に改善した事

立ちふさがった壁

1. ビルドインスタンスの進捗が把握しづらい
2. クラウドと社内のJenkins連携
3. 在宅リモートワークによるダウンロード時間
4. インスタンスへのリモートデスクトップ
5. **停止中インスタンスのEBS**
6. 複数環境でのJenkinsアカウント管理
7. ハイスペックインスタンスの動作していないCPU

EBSボリュームタイプ変更

- gp2からgp3へ変更
  - デフォルトでgp2で用意していた
- 20%程度安くなる
- 比較したところ速度に差は感じない

ビルドPCのワーカー化

- 同時に動作しているビルドPCは多くても6台
- ビルドPCは共有できるのでは？
- JenkinsからビルドPCの状態が調べることができれば…

運用中に改善した事

立ちふさがった壁

1. ビルドインスタンスの進捗が把握しづらい
2. クラウドと社内のJenkins連携
3. 在宅リモートワークによるダウンロード時間
4. インスタンスへのリモートデスクトップ
5. ~~停止中インスタンスのEBS~~
6. **複数環境でのJenkinsアカウント管理**
7. ハイスペックインスタンスの動作していないCPU

Amazon Cognitoによるユーザー管理

- Amazon Cognito
- OpenID Connect
- Jenkins プラグイン「OpenId Connect Authentication」

運用中に改善した事

立ちふさがった壁

1. ビルドインスタンスの進捗が把握しづらい
2. クラウドと社内のJenkins連携
3. 在宅リモートワークによるダウンロード時間
4. インスタンスへのリモートデスクトップ
5. ~~停止中インスタンスのEBS~~
6. ~~複数環境でのJenkinsアカウント管理~~
7. **ハイスペックインスタンスの動作していないCPU**

Incredibuild Cloudの導入

- CPUを100%使い切るのはコンパイル
- スポットインスタンスで動作させることができる
- インスタンスタイプとライセンス料のバランス

運用中に改善した事

立ちふさがった壁

1. ビルドインスタンスの進捗が把握しづらい
2. クラウドと社内のJenkins連携
3. 在宅リモートワークによるダウンロード時間
4. インスタンスへのリモートデスクトップ
5. 停止中インスタンスのEBS
6. 複数環境でのJenkinsアカウント管理
7. ハイスペックインスタンスの動作していないCPU

まとめ

社内LANに閉じてた開発を(一部)クラウドに移行することができた

- 社内分散ビルドとほぼ変わらないビルド時間
- Perforceの取得はP4P(キャッシュ)を使えば変わらない
- 今まで通りJenkinsとWindowsを使えればOK

クラウドのメリット

- インスタンス追加の手軽さ
- 物理Jenkins管理からの手放
- アクセスコントロール

改善は続く・・・

- まだ問題は残っている
- クラウドだからというものでもない

ご清聴ありがとうございました

Q & A

BLUE PROTOCOL™

To save the world that is going to destroy, fight beyond the spacetime. Cooperate with friends, beat the mighty enemies, change the history. That is your mission. Now, let's run out! On a vast land, heading for a hopeful future!

blue-protocol.com