Immutable Infrastructure を利用したソフトウェア工学教育のためのサーバ運用手法の検討

井垣　宏[†1]　福安直樹[†2]　楠本真二[†1]

SDPBL(Software Development Project-based Learning) 等の高度なソフトウェア工学教育を実施する際には，多様なサービスを導入したサーバの構築・運用を教員が実施しなければならない．本稿ではそのようなサーバ運用における課題を整理し，新しいサーバ運用手法について検討を行う．

Server Management for Software Engineering Education using Immutable Infrastructure

Hiroshi Igaki,[†1] Naoki Fukuyasu[†2] and Shinji Kusumoto[†1]

In this paper, we propose new server management process for software engineering education using a concept of “Immutable Infrastructure”.

1. はじめに

多様なソフトウェア開発技術を実際に体験し，習得することを目的としたプロジェクト形式でのソフトウェア開発演習が高度なソフトウェア工学教育の一環として多くの教育機関で行われるようになってきている．このような形式の演習をソフトウェア開発 PBL（SDPBL）と呼ぶ．

SDPBL では，プロジェクト運用や成績評価等の目的で，版管理システムやプロジェクト管理ツール等，多くのサービスを運用・管理する必要がある．本稿では，これらの運用時における課題を整理し，運用管理手法の検討を行う．

2. SDPBL 実施におけるサービス運用

SDPBL は，プロジェクトで求められる各種スキル等を学習する基礎知識学習フェーズ，プロジェクトを実施するプロジェクトフェーズ，プロジェクトを教員が評価する評価フェーズから通常構成される．

我々は文部科学省プロジェクト「分野・地域を越えた実践的情報教育協働ネットワーク（通称 enPiT)」の拠点の一つである CloudSpiral において実際に SDPBL を含むカリキュラムを実施した．以降では，CloudSpiral を例に，SDPBL において求められるサービス運用とその課題を整理する．なお，各サービスの詳細については省略する．

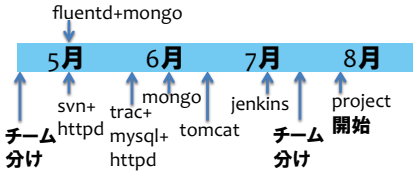

図 1　SDPBL 実施におけるサービス運用例

図 1 は 2013 年度に我々が実施した SDPBL において，サーバ環境の変更を伴うイベント及び追加されたサービスを示したものである．“fluentd+mongo” は教員が受講生の振る舞いをモニタするために導入したサービスで，それ以外はすべて受講生が利用するサービスあるいは受講生に関連するイベントである．

例えば 5 月の最初にはチーム分けが行われている．ここで我々は，チームごとのサーバに ssh や ntp 等の基本的なサービスを導入し，ユーザアカウント登録を行った．その後，実施する講義・演習に合わせて版管理システム (subversion) やプロジェクト管理ツール (trac) 等を各チームサーバに導入した．7 月の再チーム分けの時点では，それまで利用したすべてのサービスが導入された新しいサーバを新しいチームごとに作成し，8 月の project 開始を迎えた．このように我々が実施した SDPBL では学習段階に応じたサービスの

†1 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†2 和歌山大学システム工学部
Faculty of Systems Engineering, Wakayama University

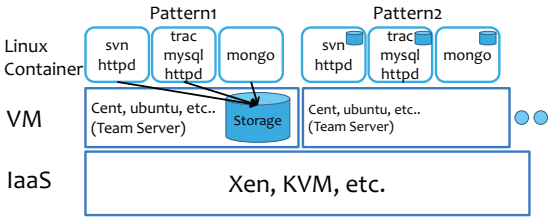

図 2 Immutable Infrastructure を利用したサーバ環境

導入，設定やデータの編集が実施された．次節ではこれらのサーバ運用時において発生した課題について詳述する．

3. サーバ運用における課題

SDPBL におけるサーバ運用において我々が直面した課題は大きく分けて以下の 3 つである．

P1. サーバ環境設定コスト サーバ環境の設定が一度の試行で完了することはほぼ無い．そのため，初期状態のサーバへの試行及びやり直しに要する時間的コストが大きな問題となった．

P2. サーバ環境設定内容の共有 複数チームのサーバに確定した設定を手分けして適用する際，その設定内容が正しく文書化されていなければならない．しかしながら，数多くのサービスが導入され，複雑化したサーバの設定を文書化することは非常に困難であった．

P3. サーバ環境のバックアップ サーバの冗長化及び過去の振り返りを目的として，バックアップが必要であったが，多くのサービスが導入されたチームサーバのバックアップは非常に困難であった．

我々は IaaS 環境を用いて環境の構築を行ったが，IaaS 環境のみでは上記の課題を解決することは困難であった．例えば，サーバ環境の設定を試行する際，VM の再起動や新規作成には数分 ~10 分程度の時間を要する．また，VM のコピーによってサーバ複製は可能であるが，複製後の変更作業が発生した場合，P2 は依然として問題である．バックアップにおいても，VM 複製のみではサーバリソースを過度に浪費してしまうため，カリキュラム終了後にはローカルにサーバ環境を再構築し，そこにバックアップを取る必要があったため，非常にコストがかかってしまった．

4. Immutable Infrastructure を利用したサーバ運用

図 2 に P1~P3 の改善を目的として我々が検討している 2 種類のサーバ環境を示す．どちらの環境も，IaaS を用いて作成されたチームごとの VM 上に導入された Linux Container[1] 上にサービスを導入する．各 Container には最低限同時に導入する必要があるサービスのみが含まれている．Pattern1 と 2 の違いは，Container 外の共通の記憶領域にデータを保持するか (Pattern1)，Container 自身がデータを保持するか (Pattern2) である．

Pattern1 では，変更の可能性がある設定及びデータが全て Container 外の共通記憶領域に保存される．これにより各 Container は初期導入完了後，不変（Immutable）なものとなる．このようなサーバ環境は Immutable Infrastructure[2] と呼ばれている．該当する Container 内の設定変更が必要になった場合は，Container を廃棄し，新たに設定を変更した Container を作りなおすことで対応する．このとき Pattern2 では，各 Container がすべての設定及びデータを保持するため，初期導入後にデータの追加等が行われていた場合，新たに Container を作成する際にデータを移行しなければならない．Pattern1 の場合は移行が必要ないため，サービスを停止することなく Container の入れ替えが可能である．

Container が Immutable になることで，サーバ環境設定内容が Container 作成時の設定と同一であることが保証される．そのため，環境設定内容の共有も Container 作成時の情報のみを共有するだけで良い．この点においては Pattern1 のほうがより Immutable であるため，利用しやすい．一方バックアップにおいては，Pattern2 の場合は Container ごとに保存するだけで良いが，Pattern1 では同時に storage を適切に配備する必要がある．この点においては Pattern2 のほうが利用しやすいといえる．両 Pattern とも，Linux Container を利用することで，VM ごとに再起動や再作成をする場合と比較して環境作成・再起動等に要する時間は飛躍的に速くなる．また，サービスごとの段階的な追加も非常に容易である．

5. おわりに

今後，Patter1,2 の環境を実際に構築し，課題がどの程度改善できているかを検証する予定である．

参考文献

1) Helsley, M.: LXC: Linux container tools, IBM devloperWorks Technical Library. (2009).
2) Chad Fowler:Trash Your Servers and Burn Your Code: Immutable Infrastructure and Disposable Components, `http://chadfowler.com/blog/2013/06/23/immutable-deployments/` (2013).