2017
秋号

百萬石

Hyakumangoku Vol.49

Published by

部長挨拶

hogas

hogas@mma.club.uec.ac.jp

「百萬石」をお手に取っていただき、ありがとうございます。弊誌 (以下、部誌) は、電気通信大学の公認サークル MMA が新歓期 (春号) と弊学が開催する調布祭時期 (秋号) の年 2 回発行・頒布しているものです。MMA とは、”Microcomputer Making Association”の頭文字に由来しており、1975 年に創立されたサークルです。現在では計算機に関連することを中心に、ネットワークやセキュリティなどの分野でも活動しています。部誌には、MMA の部員が各々に行なっている活動を記事にしたものが掲載されています。内容は多岐に渡り、高度なプログラミングの話から旅行記など、MMA の活動の自由さが感じられるものとなっております。

待ちに待った秋ですが、過ごしやすいというより、近年多くなった、三寒四温の親玉のような安定しない天候が続いているように思います。とはいえ、天気予報はほとんど外れなくなりました。天候ではありませんが、南海と呼ばれる地域の地震予測もよく耳に入るようになりました。いずれも完全な予知はできないものの、推し測る技術は進化しているようです。

最近話題の人工知能も、そのひとつとして挙げられます。データを解析し、人間のように学びつつ演算を行うため、人間の作業を肩代わりできると期待されています。かかる有名な問題として、人間が就く職業が無くなる懸念はよく聞かれますが、「人工知能がした仕事の報酬はどうなるのか」という議論もされています。「機械は文句を言わない」というのが通説でしたから、もちろん報酬など必要ないだろうという意見もあります。ただ「人間の感情のようなものが生まれるのではないか」などの見方もあり、結論に達するまでには時間がかかるでしょう。

人間と人工知能の進む道を、人間と人工知能は予測できるでしょうか？ いずれにしても平和的であってほしいものですし、私達が平和的にしていかねばなりません。

2017 年 11 月吉日

目次

第 1 章　RSA 入門　matsurika_1226　1
1.1　はじめに 1
1.2　さまざまな認証方式 1
1.3　RSA 暗号 1
1.4　脆弱性の話 (少し) 4
参考文献 4

第 2 章　スイス・フランス旅行記　n4nu　5
2.1　スイス編 5
2.2　フランス編 8
2.3　おまけ 10

第 3 章　Real World Troubleshooting　kyontan　14
3.1　はじめに 14
3.2　実例 14
3.3　まとめ 19

第 4 章　自宅サーバラックの勧め Kubernetes 編　h_otter　20
4.1　アーキテクチャ 21
4.2　ingress + kube-lego 22
4.3　まとめ 25

第 1 章

RSA 入門

matsurika_1226

1.1 はじめに

ここでは，主に RSA 暗号のアルゴリズムについて説明し，及び復号の正当性の証明を行います．また，RSA 暗号を用いるものを含む認証方式，RSA 暗号方式の脆弱性にも触れていきます．

1.2 さまざまな認証方式

RSA 暗号について説明する前に，いくつか認証方式について説明しようと思います．

理解しやすいものとして，パスワード認証方式というものがあります．サーバ側で予め登録したパスワードを暗号化したものを持っておき，ログイン時に入力されたパスワードを暗号化したものと比較します．等しければ認証し，等しくなければ認証しないという方式です．分かりやすいですね．

他には，共通鍵認証というものもあります．A 氏と B 氏のあいだで秘密にしたい情報を交換するとき，お互い同じ鍵を用いて暗号化・復号するものです．鍵を渡すときに注意が必要で，鍵が外に漏れたら意味がなくなってしまうので，悲しい思いをします．

RSA 暗号というのは，公開鍵認証に用いられる暗号方式です．RSA 暗号は公開鍵暗号の一種であるとも言えます．公開鍵認証方式では，ユーザは鍵を 2 つ持ち，それぞれを秘密鍵と公開鍵といいます．秘密鍵は秘密なのでユーザ側で隠匿しておき，公開鍵は認証されたいサーバに何かしらの方法で置いておきます．秘密鍵を公開鍵を照らし合わせることで認証を行います．

1.3 RSA 暗号

1.3.1 アルゴリズム

鍵生成

RSA 暗号の鍵は以下のようにして生成されます．

1. 素数 $p, q (p \neq q)$ を生成し，$n = pq$ とする．
2. e を $\phi(n) = (p-1)(q-1)$ 未満で，かつ $\phi(n)$ と互いに素な正の整数とする．

3. d を $\phi(n)$ を法とした e の逆元，すなわち $ed \equiv 1 \pmod{\phi(n)}$ で定める．

このとき，n と e が公開鍵，d が秘密鍵となります．p と q が分かると秘密鍵 d を求めることができてしまうので，破棄してしまいましょう．

なんだか，手順だけ見てもよく分かりませんね．そもそもこれは入門と題しているので，もう少し分かりやすく解説するべきでしょう．ということで，ひとつひとつの手順を順に追っていきます．

最初は良いですね．2 つの素数を生成し，掛け合わせて n を求めます．

手順の 2 について，$\phi(n)$ はオイラーの関数と呼ばれるもので，1 から n の自然数のうちで n と互いに素であるものの個数を示しています．ここでは単に $\phi(n)$ からは元の p, q を得にくいという理由から使われています．

手順 3 ですが，少し分かりにくいという方もおられるのではないでしょうか．この場合の逆元とは，ふつうの整数の場合とは異なるものです．$d = 1/e$ ではありません．どのようにして得るかは上に書いてあります．そういうものだとして捉えてください．

以上が鍵生成の手順です．一般に e には 65537 という値がよく使われるそうです．

暗号化

暗号化は c は平文 $m(m > n)$ に対して，以下の計算により行われます．

$$c = m^e \bmod n \tag{1.1}$$

復号

対して，復号は以下の計算により行われます．

$$m = c^d \bmod n \tag{1.2}$$

それでは，一回鍵生成から復号までを行ってみましょう．

1.3.2 鍵生成から復号までの例

鍵生成

上記の手順に従って考えてみましょう．簡単のため，$p = 3$，$q = 5$ として考えます．3 と 5 は互いに素な整数で間違いないですね．

1. 与えられた p, q より，$n = pq = 15$ が得られる．
2. $\phi(n) = (p-1)(q-1) = 2 \times 4 = 8$ となるから，e を 7 としてみる．
3. $de \equiv 1 \pmod 8$ となる d を求める．たとえば，$7 \times 7 = 49$ は 8 で割ると 1 余るので，8 を法とした e の逆数として $d = 7$ が得られる．

つまり，ここでは $n = 15$ と $e = 7$ が公開鍵，$d = 7$ が秘密鍵となります．ここでお分かりになると思うのですが，p と q に雑な値を選ぶと簡単に類推できてしまいます．そのため，実際には非常に大きな素数を p 及び q として選ぶことになります．

暗号化

それでは，平文 $m = 2$ として復号してみましょう．式 (1.1) に従うと，暗号文は

$$c = 2^7 \bmod 15 = 128 \bmod 15 = 8$$

となります．

復号

それでは，復号してみましょう．これも式 (1.2) に従うだけです．

$$m = 8^7 \bmod 15 = 2097152 \bmod 15 = 2$$

定めた平文と一致したので，無事に復号できたようです．とてもシンプルですね．しかし，なぜこのようにして復号できるのでしょうか．次の項で証明してみます．

1.3.3 証明

フェルマーの小定理によると，p を素数とし，a と p が互いに素な整数であるとき，次の式が成り立ちます．

$$a^{p-1} \equiv_p 1 \tag{1.3}$$

これを拡張して，次の式を得ることができます．

$$a^{\phi(p)} \equiv_p 1 \tag{1.4}$$

上の 2 式についてここで証明は行いません．気になる場合は各々で調べてみてください．

さて，証明にうつりましょう．暗号化の式 (1.1) の c を代入すると，復号の式 (1.2) の右辺は次のように書くことができます．

$$(m^e \bmod n)^d \bmod n \tag{1.5}$$

ところで，冪剰余の性質より，式 (1.5) は次の式と等しいことが分かります．

$$\begin{aligned} &((m^e)^d \bmod n) \bmod n \\ &= m^{ed} \bmod n \end{aligned} \tag{1.6}$$

ここで，d の定義より，$ed \equiv_{\phi(n)} 1$ なので，これを任意の整数 $l(\neq 0)$ を用いてこれを変形すると，

$$ed = l \cdot \phi(n) + 1 \tag{1.7}$$

となります．これを m^{ed} に代入すると，以下が示されます．

$$\begin{aligned} m^{ed} &= m^{l \cdot \phi(n)+1} \\ &= m \cdot (m^{\phi(n)})^l \end{aligned} \tag{1.8}$$

この式 (1.8) について，次の 2 つの場合を考えてみましょう．

1. m, n が互いに素である場合
2. m, n が共通の素因数を持つ場合

m, n が互いに素であるとき，定理より式 (1.4) が成り立つことが分かっているので，$m^{\phi(n)} \equiv_n 1$ が成立します．すなわち，式 (1.8) から順に追っていくと，

$$\begin{aligned} & m \cdot (m^{\phi(n)})^l \bmod n \\ = & m \cdot (m^{\phi(n)} \bmod n)^l \bmod n \\ = & m \cdot 1^l \bmod n \\ = & m \bmod n \end{aligned} \tag{1.9}$$

と変形できます．ここで，m は n より小さな整数なので，$m \bmod n = m$ となります．これで，(右辺) $= m$ となり，復号の式が正しいことを示せました．

次に m, n が共通の素因数を持つときについて示してみます．$n = pq$ であり，p と q はいずれも素数なので，m は p か q を素因数に持ちます．p を素因数に持たない場合について考えると，p と m が互いに素であることを用いて，フェルマーの小定理 (式 (1.3)) より $m^{p-1} \equiv_p 1$ が成り立ちます．すなわち，同様に式 (1.8) から順を追って変形していくと，下のようになります．

$$\begin{aligned} & m \cdot (m^{\phi(n)})^l \bmod n \\ = & m \cdot (m^{(p-1)(q-1)})^l \bmod n \\ = & m \cdot (m^{(p-1)})^{l(q-1)} \bmod n \\ = & m \cdot (m^{(p-1)}) \bmod n)^{l(q-1)} \bmod n \\ = & m \cdot 1^{l(q-1)} \bmod n \\ = & m \bmod n = m \end{aligned} \tag{1.10}$$

ということで，また (右辺) $= m$ が示せました．p を素因数に持つ場合は，上の p を q に置き換えればよいので，同様に成り立ちます．

以上より，RSA 暗号が正しく復号できることが示せました．最高ですね．

1.4　脆弱性の話 (少し)

最後に，RSA 暗号方式の脆弱性の話をします．平文が小さいとき，容易に暗号文から平文が想像できたり，選択暗号文攻撃という攻撃により平文を得られてしまったするという脆弱性があります．そこで，実際に用いられるときは適切な処理を加え，これらの脆弱性を回避しています．CTF で出題される際，問題はこのような脆弱性をつくものになります．

参考文献

[1] 尾関和彦 (2004)『情報技術のための離散系数学入門』共立出版.
[2]「RSA 暗号 - Wikipedia」，[online]https://ja.wikipedia.org/wiki/RSA 暗号 (参照 2017-11-22).

第 2 章

スイス・フランス旅行記

n4nu

概要
スイスとフランスの観光のついでに Google オフィスで CTF してきた.

2.1　スイス編

チューリッヒ空港から特急に乗ってチューリッヒ中央駅の広場へ.

図 2.1　チューリッヒ中央駅

一旦ホテルに荷物をおいた後，市内を観光.
左に見える時計台の建物がフラウミュンスター，右の時計台が聖ペーター教会.
手前のリマト川はチューリッヒ湖へと流れている.

図 2.2　フラウミュンスターと聖ペーター教会

チューリッヒのランドマークとも言えるグロスミュンスター

図 2.3　グロスミュンスター

リマト川に居たスイスのトリ

図 2.4　スイスのトリ

チューリッヒ中央駅まで歩いて行き，スイス国立博物館を見学

図 2.5　スイス国立博物館

チューリッヒ湖畔より

図 2.6　チューリッヒ湖

2.2　フランス編

シャルル・ド・ゴール空港からパリ市内へ出た後，サクレ・クール寺院へ

図 2.7　サクレ・クール寺院

サクレ・クール寺院のあるモンマルトルの丘から見たパリ市内

図 2.8　モンマルトルの丘から見たパリ市内

電車を使って凱旋門へ

図 2.9　凱旋門

エッフェル塔へ．日が落ちてライトアップされている
空港へ向かう直前にギャルリ・ヴェロ＝ドダへ

図 2.10　エッフェル塔

図 2.11　ギャルリ・ヴェロ＝ドダ

2.3　おまけ

図 2.12　Google 社屋

図 2.13 ガラスのドロイド君

図 2.14 Google Maps のマーカー

図 2.15　レゴの壁

図 2.16　壁

下の 2 つは CTF に問題として出された

図 2.17 スロットマシーン

図 2.18 南京錠とピッキングツール

第 3 章

Real World Troubleshooting

kyontan

3.1 はじめに

わたしと一緒に、問題解決しませんか? こんにちは、kyontan です。驚くべきことに、我々の暮らすこの世界には大小様々な問題があり、それを解決するための能力、つまり問題解決能力を問われる機会が増えつつあります。

問題解決、トラブルシューティングと呼ばれることもありますが、実社会ではしばしば問題が起こることから、これに対処する能力を学生でも養うことが重要だと考えられつつあります。[要出典]

ところで、最近の MMA の部員の活動としては ICT トラブルシューティングコンテスト[*1]への参加がありました。この大会は、いわゆるインフラ技術を対象とした大会で、ネットワークからサーバ運用, 管理, アプリケーションまで、幅広い分野でのトラブルシューティングに挑戦する大会です。MMA の部員は、このコンテストに運営や参加者として関わっており、特に参加者として参加した第 7 回, 第 8 回の大会ではどちらも優勝という成績を残しています。私は運営委員として 4 回にわたり関わっていましたが、インフラのトラブルシューティングという曖昧な範囲の中で出題される問題は幅が広く、運営であっても全部の問題を 1 人で解くことは難しいのではないかと感じています。

閑話休題ですが、今回は、更に広い文脈で、リアルワールドでのトラブル事例と、その解決策についていくつか提示します。

3.2 実例

3.2.1 MacBook Air にコーヒー牛乳を掛けてしまった

聖典にして原点。[*2] 全てのトラブルはこのように予期せずして起こるものです。

コーヒー牛乳を MacBook Air の近くで飲まないというのは根本的な解決策の 1 つですが、このような事態が起きてしまった場合どうするべきでしょうか?

まずは電源を素早く落とし、可能であれば、できるだけ早く無水エタノール等で洗い流すべきでしょ

[*1] http://www.icttoracon.net

[*2] https://twitter.com/kakakaya/status/521503975112065024

う。コーヒー牛乳が電解質かどうかは分かりませんが [要調査]、電解質であれば基板上の電極間にコーヒー牛乳が付着することで、ショートの原因となります。特に大電流が流れる電源回路周辺に付着すると大変です。その後は素早く Genius Bar で修理を依頼すべきでしょう。ただし、MacBook Air (Early 2015) 移行のモデルでは、ストレージが基盤に直接半田付けされており、故障の際は基本的にロジックボード (マザーボード) ごとの交換となってしまうため、データの復旧は絶望的です。Mid 2014 以前のモデルであれば、SSD は取り外しが可能であるため、ロジックボードのみの交換で済む可能性があります。

3.2.2 MacBook Air に梅酒を掛けてしまった

歴史は再び繰り返す。[*3] このように、稀にしか起きないと考えている出来事であっても、再び生じてパニックに陥ることがあります。

この時は、前回の知見を活かし、即座に無水エタノールを掛けて流すという対処を取りました。これにより、ダメージが最小限に抑えられたことが功を奏したのか、幸いにしてデータを復旧することができました。このように、歴史や経験から学ぶことは非常に重要です。

3.2.3 MacBook Air に白ぶどうジュースを掛けてしまった

詳細は不明ですが、現状ではまだ問題なく動作しているようです。

3.2.4 電源を入れたら破裂音がして焦げた臭いが漂ってきた

これは数年前に起きた事例です。デスクトップ PC の電源を入れたと思いきや、即座に突然の破裂音、そして焦げ臭い香りが漂ってきました。

この時の対処は、焦げ臭い香りから即座に原因が埃にあると判断し、電源周りの問題も疑い即座にコンセントを抜くというものでした。十分な時間が経過したのち、内部を分解してみると、HDD の基盤に焦げた痕跡と穴が開いている事が確認できました。見た目からは確認できませんでしたが、おそらく埃によるショートであると考えています。

予防策としては、やはり適度な間隔での掃除が上げられるでしょうか。PC の中を掃除することはなかなかないかと思いますが、このような事例が存在する以上、こまめな掃除もデータを救うことに一役買うことがあるかもしれません。

また、ストレージ単価が安くなった今の時代であれば、RAID 等の冗長化や、クラウドストレージへのバックアップも良い選択肢として挙げられるでしょう。ちなみに MMA には、無限ストレージ部というグループが存在します。キャッチフレーズは「無限の彼方へ、さあ行こう」です。

ところでこの HDD からのデータの復旧は未だにできていません。宝くじを当てたら復旧したいです。

[*3] http://twitter.com/retasu0/status/801734520403697664

3.2.5 とても分かりやすいパスワードのサーバをインターネットに公開してしまった

ユーザ名が admin でパスワードが password の VM をインターネットに公開したらどうなるのでしょうか? 実験としてであればハニーポットのような題材にもなるかもしれないですが、これが実運用されているサーバーだとしたらどうでしょうか。破滅します。

詳細は記載できませんが、 ps -ax コマンドを叩いたところ、見覚えのないバイナリが動いていることが確認できました。どうやらボットネットに感染していたようです。

即座にサーバ (本事例では VM でした) をインターネットから切断し、影響範囲の特定、具体的な影響の調査を行い、然るべき対応を行いました。

ここから得られる教訓は分かりやすいものです。インターネット上からアクセスできるサーバを簡単な認証情報でアクセスできるようにするのは絶対に行うべきではないということです。幸いにも、これによる影響は他のサーバにはなく、サービスの提供を続けて行うことができました。しかし、このサーバが例えば内部ネットワークに接続する第 2 のインタフェースを持っていたらどうなっていたでしょうか。そう考えるとゾッとしますね。

3.2.6 .env を git add して GitHub に公開してしまった

これは .env 意外にも適用できる話ですが、しばしば外部サービスの API キーなどの秘匿情報を開発用にファイルに書き込んで用いるケースはあるでしょう。こういった時に、誤ってそのファイルを git リポジトリに追加しており、あやまってインターネットに公開してしまったらどうなるでしょうか。有名な事例としては、AWS の IAM 認証情報を公開してしまったがために、300 万円分のリソースを不正利用された例がありました。[*4] 幸いにも、このケースでは AWS は不正利用された場合の請求の一部免除がありましたが、例えば顧客情報の DB の認証情報が漏れたらどうなるでしょうか。1 つ前の事例と同等の恐ろしさがありますね。

こういったときには、リポジトリを即座に private にすることは既に git clone などをしたユーザもいることから、あまり効果がありません。(もちろん、全ての問題が解決したあとには filter-branch で痕跡を消すことは良い行動であると思います) ただし、既に漏れてしまった秘匿情報の無効化が先決です。

現実問題としては、機密情報の DB は認証情報が漏れただけでアクセスできるようになるべきではありません。このような情報はインターネットから用意にアクセスできるようになっているべきではなく、イントラネット等の閉じたネットワーク内からしかアクセス出来ないようになっているべきです。

しかし、世の中の全ての情報がそのようになっているとは限らず、上記のようにリソースを不正利用されたり、情報を盗み見られるということはおうおうにして考えられます。また、追跡しようにもログの改ざん等が行われてしまった場合には、影響範囲の特定も困難になると考えられます。こういった問題に対処することは用意ではありませんが、基本的な考え方は上で述べたような多層防御、そして細めな権限の制御、そして監査であると考えます。多層防御により、本当に機密にすべき情報へのアクセスを容易

[*4] https://qiita.com/AkiyoshiOkano/items/72002409e3be9215ae7e

に行わせないようにしていれば、安直なミスによる情報漏えいを防ぐことができる可能性が高まります。また、認証情報を無効化できる機構があれば、問題の発覚時に即座にその認証情報を無効化できるでしょう。そしてどの操作がいつ行われたのか、犯人は何かといったことを追跡するためにも監査はとても重要です。世の中には、SNS サービスの社員が退職直前にそのサービスの著名ユーザのアカウントを停止することもあります。このような問題を追跡するためにはログを含めた監査機構が不可欠であると考えます。

3.2.7 cd src を打とうとして c src を打ってしまったがためにデータをロストした

事例のタイトルだけではおそらく意味がわからないでしょう。情報提供をしてくれた知人によると、この環境では c というコマンドは git checkout にエイリアスされていました。つまり、 c src というコマンドの実行は、git checkout src の実行と等価ということになります。git checkout サブコマンドは少々複雑な振る舞いを刷るコマンドです。基本的にはブランチの切り替えに利用することが多いコマンドかと思いますが、実際に存在するディレクトリ名を指定した場合には、そのディレクトリの中をコミット時にまで戻す挙動をします。つまり、これにより知人は src ディレクトリ内の全ての変更を失い、結果として src ディレクトリの中は最後のコミット時まで巻き戻ってしまったのです。南無。

教訓としては、こまめにコミットをすること、打ち間違えやすいエイリアスを設定しないこと、データの削除のような操作には確認機構や復元機構を実装することが挙げられます。

3.2.8 起動中の MySQL の DB をコピーするのに /var/lib/mysql をコピーする

稼働中の Web アプリケーションの DB である MySQL のデータをバックアップするために /var/lib/mysql をコピーするというのは正しいオペレーションでしょうか? 答えはいいえです。例えば ZFS のような CoW (Copy on Write) なファイルシステムであっても、複数ファイルを用いるデータベースのデータを単にコピーした場合には、全体での一貫性は保証されません。世の中には様々なデータベースがありますが、基本的になんらかのダンプ・リストアの手段を提供していることが多いです。この場合では、 mysqldump を用いるようにしましょう。しかし、単一ファイルの SQLite3 であれば CoW のファイルシステム上であればファイルコピーがそのままバックアップになるのではないかと筆者は想像します。

3.2.9 両親の喧嘩を止めにいったら火に油を注いでしまって離婚寸前まで追い込んでしまった

実世界の人間関係はとても複雑なものです。これは知人の実例ですが、このようなことも生きている上ではいつ起こるか分からないものです。このケースにおける知人の対応は「我々を養っていかないといけないという意識を親に植え付ける」でしたが、人間関係は特に複雑でケースバイケースの対応が求められます。やっていきましょう。

3.2.10 UPS のシリアルコンソールポートにシリアルケーブルを指したら UPS が破滅した

シリアルコンソールポートにコンソールケーブルを刺して壊れることがあるのでしょうか? このケースはそんなことはないという暗黙の思い込みを覆す良い実例です。いわゆる業務用ネットワーク機器などの管理用に用意されているシリアルコンソールポートはシリアルケーブルを PC 等と接続することで、管理コンソールへのアクセスを行うことができるようになっています。

UPS(無停電電源装置) にも同様のポートが用意されていることがありますが、この事例では稼働中の UPS のシリアルコンソールポートにシリアルケーブルを接続したところ、UPS の電源が突然落ちてしまいました。

これは設計上の仕様によるもので、シリアルコンソールポートへの接続は非通電時にのみ行われることを想定されていたことにより起こった事例です。

UPS の役割は、その名の通り接続されているデバイスへの電源供給を停電時にも安定して行うためのものです。そのような装置の管理コンソールにアクセスするためにケーブルを接続して電源が落ちるというのは、なかなか予期しづらい問題であり、またこの問題が起きたときには接続されている装置への電源供給が途絶えるわけですから、サービスの障害を起こしかねません。

対処策としては、事前にマニュアル等で装置の仕様を把握することや、稼働中のハードウェアへの操作をできるかぎり控えること、事前に様々な操作を検証することなどが挙げられます。

また、同様の挙動は某社のネットワークスイッチでも再現したことがあります。ご参考までに。

3.2.11 プリンタで印刷をしているにも関わらず白紙が排出される

目先、部誌を書いている現在 (2017/11/23 21:10) において生じている問題です。解決されたい。

この部誌をあなたが読んでいるということは、つまり問題は解決したということです。Yay!

3.2.12 朝起きられない

大変深刻な問題です。まずは以下の選択肢を検討しましょう。

- 心療内科に行く
- 精神科に行く

3.2.13 メールに気付かず面接をすっぽかす

大変深刻な問題であり、自分の将来を左右することにも繋がりかねません。以下のような選択肢を検討しましょう。

- 謝る

- 自分が起きられない時間に面談をセットしない
- 面談をしない

3.3 まとめ

さて、本記事では日常で起こりうる諸問題を取り上げ、簡潔ではありますがその解決法について述べてきました。実際には上で述べたような特定の問題というのは稀によくある程度の頻度でしか起こりません。我々の生きる実世界、実社会というのは我々が思っているよりも複雑で、そこで起こる問題には我々の想像を超えるものも多々あります。

ところで、本記事では起きた問題に対する対処法、つまり問題を解決する部分のみに絞って述べて来ましたが、実際にはそれを事前に予期し、予防することができればそれに越したことはありません。

幅広い知識を学び、そしてそれを活かし、起こりうる問題に対してその深刻度や影響範囲を予想して事前に対処することができるようになる、これは理想的な状態です。

3.3.1 SRE 本について

少し本題とは異なりますが、最近読んでいる本の中で興味深いものがあり、紹介させて頂ければと思います。「SRE サイトリライアビリティエンジニアリング」という、そのままのタイトルですが、これは Google における全てのサービスの信頼性を保つことが目的の SRE (Site Reliability Engineer) という職について、その理念や実践について、ケーススタディも合わせて紹介している本です。その中では、とあるトラブルについて以下のように述べられています。婉曲的で少し分かりづらいですが……。

> 人生は芸術 (あるいはこの場合は科学) を模倣するものであり... 不運で不可避の機会が現実の人生に生じるであろうことは想像に難くありません

この本では、どんな問題も不可避であり、最悪の事態は最悪のタイミングにおいて生じることを想定ということの重要性について再三主張されています。問題解決という文脈においても、問題に対処するスキルを磨くだけではなく、その問題の原因の追求や、それを未然に防ぐためには、といったことをを常に意識して立ち振る舞う姿勢を見習うべきだと考えます。

また、この本は単純に SRE という概念についての理解だけではなく、サービスを支えるインフラをスケールするようにするために、考えうる様々な観点からの考察が述べられています。

SRE といえば近年日本でもバズワードとなりつつある用語ですが、本当の SRE がどうあるべきかというのはあまり議論されていないように見受けられます。歴史に学ぶというのは物事を学ぶ上で欠かせない方法ですし、SRE とはなんぞや? という方も、インターネットでサービスを提供するその基盤技術の全容を眺めたいという方にも、是非おすすめしたい一冊です。日本語の訳書はオライリーから出版されていますが、英語の原著は Web で無償で公開されています。[*5]

[*5] http://landing.google.com/sre/book.html

第 4 章

自宅サーバラックの勧め　Kubernetes 編

h_otter

自宅サーバラックは非常に素晴らしいものである。どんなサーバやスイッチでもスマートに収納してくれる包容力、冬に部屋を温めてくれる安心感、そして漢のロマンを燻る無骨さが魅力であるだろう。

自宅のラックは最近以下のようになっている。

図 4.1　自宅サーバラックの様子

ToR: Cisco Catalyst 3750G
Router: NEC IX2105
Servers:
- Dell R610
- Dell R620
- 自作サーバ * 2

PDU: APC AP7900B
UPC: APC SMT1500RMJ2U
AP: Cisco Aironet 3502I

ロマンの塊である。今回は現在サーバに載せてある Kubernetes の便利さについて書いていこうと思う。

4.1 アーキテクチャ

Kubernetes は Docker オーケストレータである。つまり、複数の物理的なサーバをクラスタリングしその上で物理的な構成を気にすることなく Docker アプリケーションを展開できる。それに加えて Docker アプリケーションを支援するような機能が実装されている。

アーキテクチャは以下の図のようになっている。

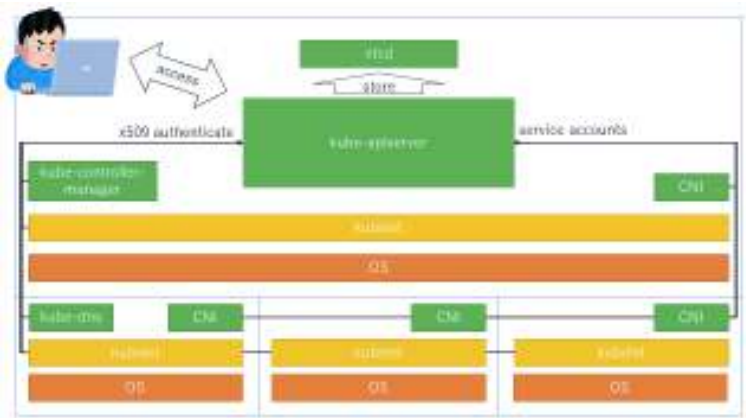

図 4.2 Kubernets のアーキテクチャ

OS の上には kubelet と呼ばれる Docker を操作するマネージャーが実行される。Kubernetes の最小構成においてワーカーと呼ばれるような実際に各ホストにインストールされるアプリケーションは kubelet のみである。もちろん Docker をインストールする必要があるが。

コントローラとしては kube-apiserver、kube-controller-manager、そして kube-apiserver のデータストアとして分散型 KVS の etcd を起動する必要がある。kube-apiserver がエンドユーザー、コントローラ、kubelet、その他クライアントアプリケーションに対する一元的な API となる。kube-controller-manager はコンテナのスケジューリングについての補助などを行っている。以上の 3 つのコントロー

ラがあれば最低限コンテナを立ち上げることができる。また、これらはすべて kubelet の manifest discovery という機能によって自動的にノードに展開を行うことができるため、とあるディレクトリに以上の 3 つのマニフェストファイルを保存することでコンテナがたちあがるのである。先程、ノードにインストールする必要があるアプリケーションが kubelet だけであると行ったのはこういう意味であり、Kubernetes は自立してクラスタリングを始めることができるのである。

実際にはこのあとコンテナのネットワークを提供する CNI の各種や、コンテナ用のキャッシュ DNS などを提供する kube-dns を入れることでようやくコンテナとして使い物になる。もちろん CNI や kube-dns も Dockernize されており、kube-apiserver にマニフェストファイルを送ることでノードを意識することなく展開をすることが可能である。

結論としては Kubernetes は比較的自立してクラスタリングを開始し、以降の自分自身の機能拡張も kube-apiserver から一元的かつ物理的な制約を感じることなく適用をすることができるアーキテクチャである。心象としては非常にスマートで拡張性が意識されており、小さくまとまっている。

認可の機能である RBAC が構築する上で一癖も二癖もあるのだが今回は省略する。

4.2 ingress + kube-lego

今回は拡張性が高い Kubernetes の中でも特に自分が便利だと思った ingress + kube-lego の組み合わせについて紹介しようと思う。

この組み合わせが便利な理由は自宅の少ない IPv4 資源を非常に効率よく一元管理することができ、TLS 証明書を発行する手間が非常に大きかったがそれを自動がすることができたため手軽にテストやサービス公開ができるようになった点である。

4.2.1 ingress とは

正直 Service との住み分けがよくわからないのだが、外部にサービスを公開するためのオブジェクトである。L7 レベルでロードバランスやアクセス制御を行うことができ、バーチャルホストや oauth2_proxy[*1] と言った認証機能などをつかうことができる。beta として現在はバージョニングされているため正式版でどうなるかは分からないが、現時点では少し機能が貧困な API ゲートウェイとして使うことができるだろう。

バックエンドには様々なものを選択することができ、Nginx や haproxy が一般的ではあるが nghttpx など様々な新しい LB も開発していたりするので色々試してみると良いだろう。今回は nginx をバックエンドとした ingress-nginx[*2] を利用したとして話をすすめる。

具体的な manifest ファイルは以下のようになる。spec.tls で TLS 証明書のホストを指定し、spec.tls[*].secretName で secret に保存した証明書を参照する。また、spec.rules でバーチャルホストとパスをサービスに関連付ける。あとは metadata.annotations で TLS 通信を使うことと HTTP 通

[*1] https://github.com/bitly/oauth2_proxy
[*2] https://github.com/kubernetes/ingress-nginx

信を HTTPS にリダイレクトすることを明示すれば Kubernetes のサービスを TLS で安全に外部へ公開することができる。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: registry
  annotations:
    kubernetes.io/tls-acme: "true"
    kubernetes.io/ingress.class: "nginx"
    ingress.kubernetes.io/ssl-redirect: "true"
spec:
  tls:
  - hosts:
    - registry.svc.example.com
    secretName: registry-tls
  rules:
  - host: registry.svc.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: registry
          servicePort: 5000
```

4.2.2 DNS について

自分としては *.svc.example.com の DNS レコードを ingress-controller の External IP に向けることをおすすめする。そのようにすると、DNS の登録を毎回することなく ingress を登録するだけで外部公開を行うことができる。更に認証機能のついていないサービスを公開する際には oauth2_proxy などで認証機能をつけることができ、後述する kube-lego との連携を行うことで Let's Encrypt[*3]の証明書を自動的に発行することで安全なサービス利用を非常に手軽に行うことができる。自宅で行っているが、テストしたい Docker アプリケーションを手軽に展開することができて非常に便利である。

External IP については Kubernetes の node にパケットを送るとよしなに Kubernetes がロードバランシングなどをしてくれる。しかし、外部から node へのバランシングや冗長化はなかなか難しい課題である。まだ試していないが、BGP の ECMP を利用した k8s-externalipcontroller[*4]というプロジェクトがあるので、今後試してみたい。現在は External IP で使うレンジをすべての node へスタティックルートを書くことで冗長化を行っている。

4.2.3 Kubernetes 管理外のサービスと ingress の連携

以下の manifest ファイルのように endpoint オブジェクトで既存の Kubernetes 管理外のサービスの IP を明示してやり、その endpoint オブジェクトを引くような service オブジェクトに作ってやれば以降は一般的な ingress オブジェクトと同様に作成することが可能である。これを利用することで VM でし

[*3] https://letsencrypt.org/

[*4] https://github.com/Mirantis/k8s-externalipcontroller/blob/master/doc/ecmp-load-balancing.md

か構築できないようなアプリケーションも含めて Kubernetes で自宅の HTTP サービスを一元管理することが可能になった。

```
apiVersion: v1
kind: Endpoints
metadata:
  annotations:
  name: gitlab
subsets:
- addresses:
  - ip: 192.168.17.4
  ports:
  - port: 80
    protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: gitlab
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  type: ClusterIP
```

4.2.4　kube-lego とは

kube-lego[*5]とは TLS の ingress オブジェクトを作成した際に、自動的に Let’s Encrypt の証明書を発行してくれるサービスである。特定のアノテーションを持った ingress オブジェクトが作成されたときにフックされるようになっており、デフォルトでは Kubernetes の ingress で TLS を使う際に明示する必要がある ‘kubernetes.io/tls-acme: ”true”‘ が指定されている。

以下の manifest ファイルのような ingress オブジェクトを作成すると自動的に証明書を発行してくれる。注意点としては secretName を指定しなければ行けない点と、kube-lego が acme チャレンジを行うために ‘/.well-known/acme-challenge‘ を ingress に登録するため競合を起こさないように特にパスを指定する必要がない場合でも ‘path: /‘ を指定しなければならない点がある。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: registry
  namespace: tools
  annotations:
    kubernetes.io/tls-acme: "true"
    kubernetes.io/ingress.class: "nginx"
    ingress.kubernetes.io/ssl-redirect: "true"
spec:
  tls:
  - hosts:
    - registry.svc.h-otter.net
```

[*5] https://github.com/jetstack/kube-lego

```
    secretName: registry-tls
  rules:
  - host: registry.svc.h-otter.net
    http:
      paths:
      - path: /
        backend:
          serviceName: registry
          servicePort: 5000
```

具体的な構築方法や kube-lego の manifest ファイルは記載しないが、気をつけなければ行けない点はいくつかある。1 つ目は RBAC を使用している場合 serviceAccount の正しい Role 定義をしなければならない。2 つめは namespace を気をつけないと行けない点である。どのオブジェクトとどのオブジェクトが同じ namespace にいなければならないかを適宜考える必要がある。もし構築中に詰まることがあれば以上の 2 点を思い出して見てほしい。

4.3 まとめ

Kubernetes は非常に便利であると思っている。物理的なハードウェアを意識することなく非常に抽象化されたサービスを利用することができる。Web サービスに偏っているので不便なところもあるがうまく OpenStack などの VM クラウド基盤と住み分けてほしい。

Kubernetes の構築で詰まったときには公式リポジトリの examples に入っている manifest ファイルを見ると非常に参考になる。更新が早いのですでに変わっている仕様や、過去のコミットで例が大幅に消されてしまうこともあるので気を付ける必要があるが、ソースコードを読んでドキュメントを全部網羅して理解するよりは非常に効率的に構築ができるだろう。

ここまで読んだ読者は忘れているかもしれない。しかし、これは “自宅サーバラックの勧め” である。もし、これを読んで便利そうだと思った人は GKE や AKE ではなく、ぜひ自宅でラックを購入しサーバ奇数台でクラスタを構築し便利な Kubernetes ライフを送ってほしい。

百萬石 2017 -秋- © 電気通信大学 MMA

2017 年 11 月 24 日　初版第一刷発行　【本書の無断転載を禁ず】
2017 年 11 月 26 日　初版第二刷発行
2017 年 12 月　5 日　第二版第一刷発行

著　者　hogas, matsurika_1226, n4nu, kyontan, h_otter
編集者　kyontan
発行者　電気通信大学 MMA
発行所　電気通信大学 MMA 部室
〒182−8585 東京都調布市調布ヶ丘 1−5−1 サークル会館 208 号室
https://www.mma.club.uec.ac.jp/
印刷所　電気通信大学 MMA 部室
製本所　電気通信大学 MMA 部室, A 棟 201 号室

電気通信大学